

Hartyányi Mária

Lengyel József

Obádovics J. Gyula

Reményi Piroska

Számítástechnika

C 64



HARTYÁNYI—LENGYEL—OBÁDOVICS—REMÉNYI

SZÁMÍTÁSTECHNIKA

A BASIC-től a gépi kódig

C 64

Novotrade · Budapest, 1989

Szerkesztette: Dr. Obádovics J. Gyula

Lektorálta: Lángos István

Varga Antal

Copyright © Hartyányi Mária, Dr. Lengyel József, Dr. Obádovics J. Gyula, Reményi Piroska, 1989

ISBN 963 585 001 8

A kiadásért felel Rényi Gábor,
a Novotrade Rt. vezérigazgatója
Budapest, 1989

Felelős szerkesztő: Sík Júlia
Készült a Szegedi Nyomdában 89-203 C 24,
Felelős vezető: Surányi Tibor igazgató

A Kiadó előszava

A számítógépek hazai megjelenése és rohamos elterjedése új távlatokat nyitott a hazai számítástechnikai alkalmazások területén. Ma már egyre inkább követelmény, hogy mind többen ismerkedjenek meg a számítógépek kezelésével, a számítógépes programok készítésével, ill. a programok használatával.

Ez a könyv bevezetést nyújt a számítástechnika alapismereteibe, a BASIC nyelv alapjától, az adatfeldolgozás módszerein keresztül, a gépi kódú programozásig.

A könyvben található programok az utóbbi években legszélesebb körben elterjedt C 64-es személyi számítógépre készültek. Ugyanakkor az egyes programozástechnikai eszközöket, adatfeldolgozási módszereket géptípustól függetlenül mutatjuk be, így a könyv azoknak is hasznos segítséget nyújt, akik más típusú személyi számítógépen dolgoznak.

1. FEJEZET

A BASIC programozási nyelv

Ebben a fejezetben a BASIC programozási nyelv Commodore 64-es személyi számítógépre alkalmazott változatát tárgyaljuk.

A BASIC a Beginner's All Purpose Symbolic Instruction Code (kezdők általános célú szimbolikus utasításkódja) elnevezés rövidítése.

A BASIC nyelvet New Hampshire-ban, a Dartmouth College-ben *Jonh G. Kemeny* és *E. Thomas* fejlesztette ki 1963—64-ben.

A BASIC az idők folyamán a mikro- és miniszámítógépek interaktív programozási nyelvévé vált. A személyi számítógépek mindegyike programozható a BASIC nyelv valamilyen változatával.

E könyv hasznos forgatásához nélkülözhetetlen egy Commodore 64-es típusú (röviden C 64-es) személyi számítógép-konfiguráció. A C 64-es számítógép alapkonzfigurációjának bekapcsolását; tv, lemezegység, központi egység sorrendben célszerű elvégezni. A konfiguráció üzemenkész állapotát — a központi egység bekapcsolása után — a tv képernyőjén a következő üzenet jelzi:

**** COMMODORE 64 BASIC V2****

64K RAM SYSTEM 38911 BASIC BYTES FREE

READY.

Ha a tv színes, akkor a képernyőn a felirat világoskék keretben, sötétkék alapon jelenik meg. A konfiguráció kikapcsolását (áramtalanítását) a bekapcsolási sorrenddel ellentétes sorrendben végezzük (központi egység, lemezegység, tv). A tápegység (transzformátor) villásdugóját is célszerű kihúzni a hálózati csatlakozóaljzatból, ha hosszabb ideig nem használjuk a számítógépet.

A képernyőn megjelenő üzenetből a felhasználó számára fontos információ, hogy a központi tárban 38 911 byte tárolókapacitás áll rendelkezésre.

A READY. (kész) üzenet alatt egy villogó négyzet, az ún. *kurzor* látható, amely azt jelzi, hogy az általunk leütött billentyűnek megfelelő karakter a képernyőn hol fog megjelenni.

A C 64-es operációs rendszere két üzemmódban fogad el BASIC kifejezéseket:

- számológép- (direkt) üzemmódban és
- tárolt programú üzemmódban.

A számológép-üzemmódban használható BASIC kifejezéseket sorszámozás nélkül írjuk be a billentyűk egymás utáni leütésével, és ezeket a

RETURN

billentyű lenyomásával azonnal végrehajtatjuk. A továbbiakban az ilyen BASIC kifejezéseket *parancsoknak* nevezzük.

Tárolt programú üzemmódban a BASIC utasításokat az utasítássor első pozíciójától kezdve sorszámozva írjuk be.

Egy utasítássor legfeljebb 80 karakterhely hosszúságú lehet, amely két képernyősornak felel meg. A BASIC értelmező (interpreter) számára az utasítássor végét a

RETURN

billentyű leütése jelzi (hatására a kurzor a következő sor elejére ugrik).

A számítógép számológép- (direkt) üzemmódban való használata, a billentyűkre vonatkozó ismeretek és egyéb alapismeretek az irodalomjegyzékben megadott könyvekből, jegyzetektől sajátíthatók el.

1.1 A BASIC NYELV ALAPJELEI

A C 64-es számítógép billentyűivel közvetlenül előállítható és a képernyőn megjeleníthető karakterek és grafikus jelek összességét a Függelék tartalmazza.

A BASIC nyelv parancsaiban és utasításaiban használható karakterek

1. Számjegyek: 1 2 3 4 5 6 7 8 9 0

2. Az angol ábécé nagy-, ill. kisbetűi:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

3. Speciális karakterek:

szóköz (blank); a kulcsszavak, változók elkülönítésére (a szóközök kitétele a C 64-es BASIC nyelvben nem kötelező. Elhagyásuk a program áttekinthetőségét rontja, de takarékossgot jelent a tárral)

; pontosvessző; változók listájában nyomtatási kép formálására

= egyenlőségjel; értékadás jele, relációs jel az egyenlőség vizsgálatához

+ pluszjel; aritmetikai összeadás és fűzér konkatenáció jele

- mínuszjel; aritmetikai kivonás és negatív mennyiség előjele

* csillag; aritmetikai szorzás jele

/ ferde törtvonal; aritmetikai osztás jele

↑ felfelé mutató nyíl; aritmetikai hatványozás jele

(bal oldali

) jobb oldali zárójel; részkifejezések, függvényargumentumok jelölésére

% százalékJel; egész típusú változók jelölésére

számjel; a beviteli-kiviteli utasításokban a logikai file-szám előtti jel

\$ dollárjel; fűzér (sztring) típusú változók jelölésére

, vessző; változók listájában nyomtatási kép formálására, utasítások paramétereinek elválasztására

. pont; lebegőpontos számokban tizedespont (tizedesvessző nem használható)

” idézőjel; fűzér (sztring) konstansok határolására

: kettőspont; sorba írt BASIC utasítások, parancsok elkülönítésére

? kérdőjel; a PRINT kulcsszó rövidítésére

< kisebb, mint ... jel; relációs jel konstansok, kifejezések, fűzések (sztringek) összehasonlítására

> nagyobb, mint ... jel; relációs jel konstansok, kifejezések, fűzések (sztringek) összehasonlítására

π pi; 3.141592654 numerikus konstans

@ kapcsolójel; lemezkezelő parancsban pl. program felülírására

4. Grafikus jelek:

  stb. (l. 2. táblázat)

A  és a  billentyűkkel (ún. váltók) állítható be a „nagybetűk/grafikus jelek”, ill. a „kisbetűs/nagybetűs” karakterkészlet.

A „nagybetűk/grafikus jelek” beállítás esetén a felső lapjukon egy jelű billentyűk leütésekor az a karakter kerül a képernyőre, amelyik a lapon látható, ill. a két jelű billentyűk esetén a lapon látható karakterek közül az alsó.

A  billentyű lenyomott állapota mellett az egy jelű billentyűk előlapján látható jobb oldali grafikus jel, ill. a két jelű billentyűkön látható karakterek közül a felső: a  billentyű lenyomott állapota mellett pedig a billentyűk előlapján látható bal oldali grafikus jel jelenik meg az adott billentyű leütése esetén.

A „kisbetűs/nagybetűs” beállítás esetén a betűbillentyűk leütésével kisbetűkkel, a  billentyű lenyomott állapota mellett pedig nagybetűkkel írhatunk a képernyőre.

A  billentyű lenyomott állapota mellett a billentyűk leütésével az elől látható bal oldali grafikus jeleket írhatjuk a képernyőre.

1.2 KONSTANSOK, VÁLTOZÓK

1.2.1 Konstansok

Konstansok a C 64-es számítógép BASIC utasításaiban használt adatértékek. A konstansok három típusa:

- egész számkonstansok;
- lebegőpontos számkonstansok;
- fűzér- (sztring) konstansok.

a) Az egész konstansok a C 64-es ábrázolhatósági feltétel szerint azok az egész számok, amelyek a $[-32768; +32767]$ intervallumba esnek.

Az egész konstans nem tartalmazhat sem tizedespontot, sem vesszőt. A pozitív egész számok előjele kiírható, de el is hagyható. Az egész számok a tárban két byte hosszúságú bináris számként tárolódnak. Pl.:

egész konstansok: 3
-145
32 766
+327
-2768
0

nem egész konstansok:

3.
-145,
-32768

Megjegyzés: a C 64-es a PRINT -145, parancsot hibajelzés nélkül (az odaírt vessző ellenére) végrehajtja és a -145 egész számot írja ki, de az A = -145, : PRINT A parancsokat nem fogadja el

? SYNTAX ERROR

üzenetet küld. Hasonlóan a PRINT 3. parancsot végrehajtva a 3 egész alakú kiírást végzi el.

Az A % = 3.45 : PRINT A % parancsok végrehajtása után a képernyőre szintén csak 3 íródik ki, hibajelzés nélkül (a % jel egész típusú változót jelöl). De az A % = 32768 : PRINT A % parancsokat

? ILLEGAL QUANTITY ERROR

üzenettel a gép nem hajtja végre (1-el nagyobb, mint a megengedett legnagyobb egész szám).

Figyelem! A számok mögé nem írhatunk % jelet a C 64-es BASIC-ben, akkor ui.

? SYNTAX ERROR

üzenetet kapunk.

Az egész számok jegyei között álló szóközöket a gép nem veszi figyelembe. Pl. az A = 1 2 : PRINT A parancsok végrehajtása után a képernyőn a 12 szám jelenik meg.

b) *A lebegőpontos konstansok* a C 64-es számára azok a negatív vagy pozitív valós számok, amelyek abszolút értékben a

$[+2.93873588 \times 10^{-39}; +1.70141183 \times 10^{+38}]$

intervallumba esnek. Az egész részt és tört részt tizedesponttal kell elválasztani. A pozitív számok előjelét nem szükséges kiírni. Ha az egész rész értéke 0, akkor a tizedes-pont előtti 0-t sem szükséges kiírni, pl. 0.45 helyett .45 írható. Ha a tört rész értéke 0, akkor a tizedespont utáni 0-kat is elhagyhatjuk, pl. 37.000 helyett 37. írható.

A valós konstans általában öt részből áll: előjelből, egész részből, tizedespontról, tört részből és kitevő részből. A kitevő rész E+n, ill. E-n alakú, ahol az n -39 $\leq$ n $\leq$ +38 intervallumba eső egy- vagy kétjegyű egész szám és 10⁺ⁿ, ill. 10⁻ⁿ tényezővel való szorzást jelent.

A valós számkonstans helyes felírásához nincs szükség minden esetben mind az öt részre, de a tizedespont és a kitevő rész közül legalább az egyiknek, ill. az egész- és a tört rész közül legalább az egyiknek szerepelnie kell. A + előjel elhagyható, az előjel nélküli valós számkonstans pozitív.

Példák helyesen felírt lebegőpontos számkonstansokra:

247. -45.020456 5.47
-.218 +93.749 .624
+2E4 2E+4 (mindkét írásmódnak megfelelő szám 20000, mivel $2 \times 10^4 = 20000$)

-7.2 E-12 (-7.2 $\times 10^{-12}$ számmal azonos)
4.3 E+1 (4.3-mal azonos)
43 E-1 (4.3-mal azonos, négy egész három tized)
43 E vagy 43E0 (43 $\times 10^0 = 43$)

Figyelem! E2 $\neq 0 \times 10^2$, a BASIC értelmező ezt az E2 nevű változóként kezeli, amíg értéket nem kap E2=0, de 1E2=100.

Az exponenciális alakban adott lebegőpontos számkonstansok E előtti részét *mantissának*, az E-vel kezdődő részét *karakterisztikának* vagy *kitevőnek* nevezzük.

A C 64-es a képernyőre legfeljebb kilenc jegyű lebegőpontos számokat ír ki exponenciális rész nélkül, ha azok a [-999999999; -0.01], ill. a [+0.01; +999999999] intervallumokba esnek. Ha a szám abszolút értékben .01-nél kisebb vagy 999999999 nagyobb, akkor az a képernyőn exponenciális alakban jelenik meg.

A lebegőpontos számok *kerekítését* a gép a tizedik jegy alapján végzi: ha a tizedik jegy 5 vagy 5-nél nagyobb, akkor az előtte levő részt felfelé, ha kisebb, akkor pedig lefelé kerekíti.

Ha az ábrázolandó szám — mint már említettük — abszolút értékben kisebb a +2.93873588E-39 számnál, akkor az értelmező hibajelzés nélkül 0.0 értékkel (gépi nulla) számol tovább, ha pedig a +1.70141183E+38 számnál nagyobb, akkor a

? OVERFLOW ERROR (túlszordulási hiba)

üzenettel megszakítja a programot.

c) *Fűzér vagy sztring konstans* alfanumerikus és/vagy grafikus karaktereket tartalmazhat. A fűzér hossza legfeljebb 256 karakter lehet, de a képernyőről csak 80 karakter hosszú sor adható át feldolgozásra. A fűzért idézőjelek között kell megadni. Maga a fűzér idézőjeleket nem tartalmazhat, de CHR\$(34) használatával a szövegben megjeleníthetjük az idézőjeleket (l. az 1.13 alfejezetben.) Pl. a FELKIAL-TOTT "HAHO" ES TOVABBMENT. szöveget a következő parancs írhatja ki a képernyőre:

```
PRINT CHR$(34)"FELKIALTOTT"CHR$(34)"HAHO"CHR$(34)"  
ES TOVABBMENT."CHR$(34)
```

Az üres fűzért egymás mellé írt idézőjelpárral adjuk meg: ""

1.2.2 Változók

A változók a BASIC utasításokban az adatok értékeit reprezentáló nevek. A konstansoknak megfelelően a változók is *egész típusúak*, *valós (lebegőpontos) típusúak* és *fűzér- vagy sztring típusúak* lehetnek.

A változók típusonként lehetnek:

- egyszerű és
- indexes, azaz tömbváltozók.

A változókat névvel látjuk el. A név % jellel egész típusú, \$ jellel fűzér típusú, e két jel hiányában pedig valós (lebegőpontos) típusú változóra utal. A változó nevét *azonosítónak* nevezzük. Az azonosítók a típusoktól függetlenül, csak betűkarakterrel kezdhetők, majd betűk és/vagy számjegyek kombinációjával folytathatók. A C 64-es BASIC a változó nevének hosszát 255 karakterben korlátozza, de a változókat csak az első két karakter alapján különbözteti meg egymástól.

A változók azonosítói, ill. azok részei nem lehetnek a BASIC kulcsszavak, függvénynevek és logikai operátornevek. Amennyiben ilyen azonosítót használunk, akkor a képernyőn a

? SYNTAX ERROR

üzenet jelenik meg.

a) Egyszerű változók az index nélküli változók.
Pl. valós típusváltozók azonosítói:

A BARLANG BABA ALFA V12B3 BA
EREDMENY C1 OSZTALY MARI BE

Az értelmező a BARLANG, BABA és BA neveket nem különbözteti meg, mindhármát BA azonosítónak kezeli.

```
BARLANG=3:PRINT BARLANG: ?BABA: ?BA
3
3
3
```

READY.

A BAGOLY nem lehet azonosító, mert a GO kulcsszórészt magában foglalja.

```
PRINT BAGOLY
0
?SYNTAX ERROR
READY.
```

Minden egész és valós típusú változó, amíg értéket nem kap, 0 értékű, a függőváltozó pedig üres függő ("") tartalmú.

Egész típusú változók azonosítói:

A% BULI% VAU% V1% V8%
CUMI% CILI% MEZ% DARAB%
Pl.:

```
AX=147:V1%=1470:DARAB%=456:PRINT AX%.V1%.DARAB%
147      1470      456
READY.
```

Figyelem! A K%=16.457: ?K% parancsok végrehajtása után hibajelzés nélkül a 16 egész szám íródik a képernyőn, mert a K% változó csak az egész részt veszi át. Hasonlóan, ha D%=.4897: PRINT D% parancsok végrehajtása után D% értéke 0; ez kerül kiírásra.

b) Indexes, azaz tömbváltozó a változó neve és típuskarakter után egy bal oldali zárójellel folytatódik, majd vesszővel elválasztott indexszekkel (számkonstanssal, ill. azonosítóval) és egy jobb oldali zárójellel fejeződik be. Pl.: P1(1, 2, 4) R(A, B, C) Z(8, 3) V(4) B%(2) D\$(7) Az indexek karakterei és a zárójelek nem részei a változó nevének.

Az indexes változó használata lehetővé teszi, hogy pl. egy 20 sort és soronként 8 adatot tartalmazó táblázat 160 adatára egyetlen indexes azonosítóval hivatkozzunk 160 különböző egyszerű azonosító helyett. A T(4, 6) tömbem a táblázat 4-es sorának 6-os elemét, a T(19, 2) pedig a 19-es sor 2-es elemét jelöli stb.

A tömb típusát az egyszerű változó típusával azonos módon jelöljük. Pl.:

V(K) — valós típusú egydimenziós tömb (vektor);

E%(J, R) — egész típusú kétdimenziós tömb (mátrix);
MA\$(I, J, K) — függő típusú háromdimenziós tömb;
T(I, J, K, L, M) — valós típusú ötdimenziós tömb.

A tömb maximális dimenziószáma elméletileg 255 és dimenzióként az elemek lehetséges száma 32767, gyakorlatilag azonban a tár kapacitása és a 80 karakter hosszúságú utasítássor szűkebb korlátot szab.

A tömbök méretét, azaz az indexek által felvehető legnagyobb értékeket a BASIC nyelv DIM utasításában kell megadni (l. az 1.10 alfejezetben). Az egy-, két- és háromdimenziós tömböket nem kell feltétlenül megadni, ha az indexek egyike sem vesz fel 10-nél nagyobb értéket a program futása során. Az értelmező ui. automatikusan 11 helyet biztosít az egydimenziós tömb, 11*11 helyet a kétdimenziós tömb és 11*11*11 helyet a háromdimenziós tömb számára, mivel az indexek értéke a 0-tól indul.

Négy vagy négynél több dimenziós tömböt mindig deklarálni kell, mert ha deklarálás nélkül előfordul ilyen tömb a programban, akkor a program futtatásakor

? OUT OF MEMORY ERROR

hibajelzéssel leáll a program.

A tömb tárolásához szükséges tárterület:

2 byte a tömb neve, 2 byte a tömb végcíme +1 és 1 byte a dimenziók számának megadására;

+2 byte a tömb minden dimenziója számára;

+2 byte az egész elemek mindegyike számára;

vagy +5 byte a lebegőpontos elemek mindegyike számára;

vagy +3 byte a függő elemek mindegyike számára;

és +1 byte a függő elemek mindegyik karakterek számára.

Az indexek lehetnek egész vagy valós konstansok, változók aritmetikai kifejezések, de az eredmény minden esetben egész értékként kerül felhasználásra. Pl.:

az A(2.5, 1, 3/2) indexes változót

A(2, 1, 1) tömbelemként használja az értelmező.

Az indexek 0-tól a meghatározott felső indexhatárig minden egész értéket felvehetnek. Negatív indexérték esetén

? ILLEGAL QUANTITY ERROR

üzenetet kapunk. Ha a deklarált maximális indexértéket túllépjük, akkor a

? BAD SUBSCRIPT ERROR

üzenet jelenik meg a képernyőn.

Példák tömbökre értékadással:

EL(Z%)=35

MA%(2*I, K+3*J)=B%(2)

KUB(D, E, F)=1986.45

RS(5)="1986.03.03"

egydimenziós lebegőpontos tömb (vektor);
két dimenziós egész típusú tömb (mátrix);
háromdimenziós lebegőpontos tömb (az F sor-
számú lap D sorának E-edik eleme);
függőtömb, ötös sorszámú (6.) eleme

1.3 MŰVELETEK, KIFEJEZÉSEK

A konstansokat és az azonosítókat, hacsak nem kívánjuk azokat egyedi tulajdonságaik alapján megnevezni, általában *operandusoknak* hívjuk.

Az 1.3.1 pontban leírt aritmetikai műveletek a BASIC utasításokban és a számológép-üzemmódban használt parancsokban azonosan használandók és prioritásuk is megegyezik.

Ebben az alfejezetben az aritmetikai műveletek mellett, a *relációs operátorok* bevezetésével, a *relációs műveleteket* és a *logikai műveleteket* is ismertetjük.

Kifejezéseknek nevezzük az operandusok és műveleti jelek olyan kombinációit, amelyek eredményül egyetlen értéket hoznak létre.

A kifejezések az operandusaik, ill. eredményük alapján lehetnek

- a) aritmetikai és
- b) füzér (sztring)

kifejezések.

1.3.1 Aritmetikai kifejezések

Az *aritmetikai kifejezés* számkonstansok, azonosítók és műveleti jelek kombinációja, amelynek kiértékelése egész vagy valós számértéket eredményez.

A C 64-es parancsaiban és utasításaiban az aritmetikai kifejezések

- aritmetikai;
- relációs és
- logikai

műveleti jeleket, ill. operátorokat és zárójeleket tartalmaznak.

Aritmetikai műveleti jelek (alpműveletek):

- + összeadás, pl. $a + b$
- kivonás, pl. $a - b$
- * szorzás, pl. $a * b$
- / osztás, pl. a / b
- ↑ hatványozás, pl. $a \uparrow b$ (a^b)

Pl. az $\frac{5(a-b^2)}{c} + \frac{d}{(a+b)^2}$ algebrai kifejezésnek megfelelő BASIC aritmetikai kifejezés:

$$5*(A-B\uparrow 2)/C+D:(A+B)\uparrow 2$$

Az alpműveleteket tartalmazó aritmetikai kifejezésekben a zárójelek felbontása, a hatványozás elvégzése és a mínusz előjel értelmezése után a szorzás és az osztás művelete hajtódik végre, az ún. „balról jobbra” szabály szerint, azt követően pedig az összeadás és a kivonás művelete, szintén a „balról-jobbra” szabály szerint.

A „balról jobbra” szabály azt jelenti, hogy a zárójelen belüli, ill. a zárójel nélküli kifejezést az értelmező a bal oldali karaktertől kezdi kiértékelni, és az egymás után következő összes szorzási és osztási műveletet rendre elvégzi, majd hasonló módon elvégzi az összeadási és a kivonási műveleteket.

Az alpműveletek elsőbbségi sorrendje (prioritása):

1. ↑ hatványozás (alap ↑ kitevő)
2. - mínusz előjel értelmezése ($-A$)
3. * / szorzás, osztás ($A*B*C/D*E$)
4. +- összeadás, kivonás ($A-B+C+D$)

Aritmetikai műveletekkel adott kifejezések:

BASIC kifejezés

$$A * X1 - (A/X1 - 5) + 23 \uparrow 3$$

$$-4 \uparrow 2 + A * B / (4 * A) / B$$

$$R \uparrow (A/B) + 5 \uparrow - 2 * R$$

$$(-B + (B * B - 4 * A * C) \uparrow .5) / (2 * A)$$

$$(V + D) \uparrow (1/7) - V * D / 2$$

$$(G \% + 1) * (A \% - 1)$$

$$A \% / B \%$$

Relációs operátorok:

- < kisebb, mint
- = egyenlő
- > nagyobb, mint
- <= kisebb vagy egyenlő
- >= nagyobb vagy egyenlő
- <>, >< nem egyenlő.

Két operandust vagy kifejezést a relációs operátorok valamelyikével összekapcsolva a reláció értéke -1 vagy 0, attól függően, hogy a reláció által kifejezett állítás igaz vagy hamis.

Pl. $5 < 6$	igaz állítás, értéke	-1
$5 > 6$	hamis állítás, értéke	0
$5 = 6$	hamis állítás, értéke	0
$5 = 8 - 3$	igaz állítás, értéke	0
$9 > (95/10 - .5)$	igaz állítás, értéke	-1
$10 < > \pi * 3$	igaz állítás, értéke	-1

Algebrai értelmezése

$$ax_1 - \left(\frac{a}{x_1} - 5 \right) + 23^3$$

$$-4^2 + \frac{ab}{4a/b}$$

$$r^{\frac{a}{b}} + \frac{1}{5^2} \cdot r$$

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$\sqrt[7]{v+d} - \frac{vd}{2}$$

$$(g+1)(a-1)$$

(g és a egész érték)

$$\frac{a}{b}, \text{ de nem „egész” osztás}$$

Pl.:

A=3.1415:B=3:Z=10<>A*B:?"Z=":Z
Z=-1

READY.

Ha a numerikus operandusok típusa eltérő, pl. az egyik egész típusú, a másik valós típusú, akkor a két operandus összehasonlítása előtt mindkét operandust lebegőpontos típusúvá alakítja az értelmező és a reláció értékét ez után állapítja meg.

Logikai operátorok:

AND logikai ÉS (szorzás);
OR logikai VAGY (összeadás);
NOT logikai NEM (tagadás, negáció).

A logikai operátorok egyrészt módosíthatják a relációk értelmét, másrészt numerikus eredményt állíthatnak elő. A logikai műveletek operandusai lehetnek konstansok, változók, relációk, aritmetikai és fűzér kifejezések. A logikai operátorok közül az AND és az OR kétváltozós művelet, a NOT pedig egyváltozós művelet.

Abban az esetben, ha az operandusok csak az igaz vagy hamis állításoknak megfelelő -1 vagy 0 értéket vehetnek fel, akkor a logikai műveletek eredménye is csak -1 vagy 0 érték lehet.

A logikai műveletek eredményei a Boole-algebra szabályai szerint bitenként végzett műveletek eredményeként állnak elő a következőkben ismertetett módon.

Kétváltozós műveletek:

$\begin{matrix} 1 \text{ AND } 1 = 1 \\ 0 \text{ AND } 1 = 0 \\ 1 \text{ AND } 0 = 0 \\ 0 \text{ AND } 0 = 0 \end{matrix}$
 Tehát az AND művelet eredménye akkor és csak akkor 1, ha mindkét bit 1 (egyébként 0).

$\begin{matrix} 1 \text{ OR } 1 = 1 \\ 0 \text{ OR } 1 = 1 \\ 1 \text{ OR } 0 = 1 \\ 0 \text{ OR } 0 = 0 \end{matrix}$
 Tehát az OR művelet eredménye akkor és csak akkor 0, ha mindkét bit 0 (egyébként 1).

Egyváltozós művelet:

NOT $1 = 0$ A NOT művelet a bit logikai komplementjét állítja elő.
NOT $0 = 1$

Az értelmező a logikai operandus értékeit a műveletek elvégzése előtt 16-bites bináris egész számmá alakítja és megvizsgálja, hogy azok a

$[-32768; +32767]$

intervallumba esnek-e, ha igen, elvégzi a kijelölt műveletet, ha nem akkor? ILLEGAL QUANTITY ERROR üzenetet ad.

Pl.:

PRINT 13 and 14
12

READY.

	Bináris 16-bites művelet	Decimális művelet
	0000 0000 0000 1101	13
AND	0000 0000 0000 1110	14
	0000 0000 0000 1100	12

PRINT 13 AND 113
1

READY.

	Bináris 16-bites művelet:	Decimális művelet
	0000 0000 0000 1101	13
AND	0000 0000 0111 0001	113
	0000 0000 0000 0001	1

PRINT 32002 AND 28763
28674

READY.

	0111 1101 0000 0010	32 002
AND	0111 0000 0101 1011	28 763
	0111 0000 0000 0010	28 674

PRINT 13 OR 14
15

READY.

	0000 0000 0000 1101	13
OR	0000 0000 0000 1110	14
	0000 0000 0000 1111	15

PRINT 44 OR 113
125

READY.

	0000 0000 0010 1100	44
OR	0000 0000 0111 0001	113
	0000 0000 0111 1101	125

PRINT 74 OR 181
255

READY.

	0000 0000 0100 1010	74
OR	0000 0000 1011 0101	181
	0000 0000 1111 1111	255

PRINT NOT 13
-14

READY.

A NOT X hatására a $-(x+1)$ kifejezésnek megfelelő érték képződik: $-(13+1) = -14$.

PRINT NOT 14, NOT 44, NOT 0, NOT -1,
-15 -45 -1 0

READY.

Az igaz ill. hamis értékű kifejezésekre alkalmazott logikai műveletek eredménye vagy 0, vagy -1 .


```
A=2>3:B=4=4:7A AND B:7A OR B:7NOT A:7NOT
B:7"A=":A:7"B=":B
0
-1
-1
0
A= 0
B=-1
```

READY.

Figyelem! A B=4=4 parancs jobb oldala a 4=4 reláció, amelynek igaz volta miatt a B értéke -1.

Az értelmező az aritmetikai műveleteket, relációs és logikai operátorokat, zárójeleket tartalmazó BASIC kifejezés kiértékelését, a műveletek elvégzésére előírt *elsőbbségi sorrend* (prioritás) szerint végzi a zárójeleken belül, ill. felbontásuk után azokon kívül is:

- | | |
|-------------------|---|
| 1. ↑ | hatványozás |
| 2. - | mínusz előjel érvényesítése |
| 3. */ | szorzás, osztás (az azonos szintű műveletek között a „balról jobbra” szabály az érvényes) |
| 4. + - | összeadás, kivonás |
| 5. > = < <= >= <> | relációs operátorok |
| 6. NOT | logikai NEM |
| 7. AND | logikai ES |
| 8. OR | logikai VAGY |

Megjegyzés: a kifejezésben a kezdő (bal oldali) zárójelek számának a befejező (jobb oldali) zárójelek számával meg kell egyeznie. Ellenkező esetben az értelmező ? SYNTAX ERROR üzenetet küld.

1.3.2 Füzer- (sztring-) kifejezések

A füzerváltozót — amint azt már említettük — a \$ jel különbözteti meg az egész és a lebegőpontos változótól, pl. A \$. Füzer változókat, ill. füzereket (idézőjelekkel közrezárt karakterek) a relációs operátorokkal ugyanúgy összehasonlíthatunk, mint a számokat, ill. az aritmetikai kifejezéseket.

A füzerek összehasonlítása karakterenként balról jobbra halad a karakterkódok alapján. Két füzer akkor egyenlő egymással, ha a karakterek száma egyenlő (beleértve a szóköz karaktereket is), valamint balról jobbra haladva a megfelelő helyeken azonos karakterek állnának.

Pl.:

```
Z$="MARI":X$="MERI":?Z$="MARI":?Z$=X$:?Z$<X$
-1
0
-1
```

READY

Amint látjuk, a Z\$="MARI", azaz a "MARI"="MARI") állítás igaz (-1); a Z\$=X\$ állítás hamis (0) és a Z\$<X\$ igaz, mivel a "MARI" és a "MERI" füzerekben a 2. karakter eltérő és az A-nak 65, az E-nek pedig 69 az ASCII kódja (l. a Függelékben).

A füzerváltozókat, ill. a füzereket a pluszjellel (+) összekapcsolhatjuk. Ezt a műveletet *konkatenációnak* nevezzük. A művelet eredményeként a konkatenáció jelének bal oldalán levő füzer és a jobb oldalán levő füzer a sorrend meghagyásával egy füzerré kapcsolódik össze.

Pl.:

```
Q$="JO ":W$="NAPOT !":R$=Q$+W$:?R$
JO NAPOT !
```

READY.

```
A$="RIA, ":B$="HUNGARIA !":C$=A$+B$:?A$+C$
RIA, RIA, HUNGARIA !
```

READY.

A *füzérkifejezés* füzérkonstansok, füzerváltozók, relációs operátorok és a konkatenáció műveleti jel kombinációja.

Az értelmező füzérkifejezések és aritmetikai kifejezések közötti műveleteket nem fogad el. Ilyenkor a

? TYPE MISMATCH ERROR

üzenetet küldi.

1.4 ÉRTÉKADÓ UTASÍTÁSOK

Az értékadó utasítások alakja:

$s \ V=k$, ill. $s \ LET \ V=k$,

ahol az s sorszám, a V tetszőleges változó azonosítója, az egyenlőségjel az értékadás jele, a k pedig egy tetszőleges kifejezés (konstans, változó, egyszerű vagy összetett aritmetikai, ill. füzer kifejezés) lehet. A LET szó használata a C 64-esen nem kötelező.

Az értékadó utasítás hatására megvalósul

- a V változó azonosítása — indexes változó esetén ez az indexkifejezés aktuális értékének kiszámítását is magában foglalja;
- a k kifejezés kiértékelése, ill. a k aktuális értékének kiszámítása;
- a k kifejezés aktuális értékének hozzárendelése a V változóhoz.

Ha V és k típusa azonos (mindkettő egész, valós vagy füzer típusú), akkor közvetlenül sor kerül az értékadásra.

Ha V valós és k egész típusú, akkor a V változó k valósra alakított értékét veszi át.
 Ha V egész típusú (V%) és k valós típusú, akkor a V% változó k egészre alakított értékét veszi át (a tizedesponttal együtt a tört rész elmarad).
 Ha V és k közül csak az egyik füzér típusú, akkor a program végrehajtása

? TYPE MISMATCH ERROR IN s

üzenettel megszakad, ahol az s a hibát tartalmazó programsor sorszáma. Pl.: ha a

10 V=A\$2+B1

utasítás végrehajtásakor a

? TYPE MISMATCH ERROR IN 10

hibaüzenettel megszakad a program végrehajtása.

1.5 A PRINT, A RUN, A REM, A LIST ÉS A NEW KULCSSZAVAK

A számológép-üzemmódban használt parancsok a sorszámozott programsorokban a BASIC nyelv utasításáivá válnak. Pl.:

```
10 PRINT "1":REM*** 1. PROGRAM
14 :
15 REM** EGYSZERU PROGRAM SZEMLELTETESE
16 :
20 A=3:REM** ERTEKADAS
30 B=2:REM** ERTEKADAS
40 C=A/B+3*8:REM** ERTEKADAS AZ ARITMETIKAI
42 REM** KIFEJEZES KISZAMITASAVAL.
50 PRINT "C=";C:REM** EREDMENY NYOMTATASA
```

A leírt utasítások, amelyek együtt a

$$c = \frac{3}{2^3} \cdot 8$$

kiszámításának programját alkotják, mindaddig megmaradnak a C 64-es operatív tárában, amíg azokat nem töröljük. (Törlés: NEW RETURN).

Az utasításokhoz a kettőspont után, vagy külön utasításorba a REM (remark) kulcsszóval megjegyzéseket írhatunk. Az értelmező a REM-mel kezdődő részt nem veszi figyelembe, a program végrehajtását a következő sortól folytatja. A REM-mel kezdődő rész tehát csak a programlistában jelenik meg.

A tárolt programot a

RUN

parancs hajtja végre. Írjuk be a RUN kulcsszót és nyomjuk le a RETURN billentyűt! Az első utasítás (10-es sorszámu) törli a képernyőt, a 20-as és a 30-as sorszámu ún. *értékkadó utasítás* az A és B valós változóknak ad értéket, így az A, ill. a B program további részében 3, ill. 2 értékkel vesz részt a kijelölt műveletben. A 40-es sorszámu utasítás az egyenlőségjel jobb oldalán álló aritmetikai kifejezését kiértékeli és az a C változóban tárolja, majd az 50-es sorszámu utasítás kinyomtatja a C=3 eredményt.

A tárből a képernyőre a

LIST

parancssal írhatjuk ki a programot. Írjuk be a LIST kulcsszót és nyomjuk meg a RETURN billentyűt! A program megjelenik a képernyőn.

A program nagyon rugalmatlan, mivel minden egyes futásakor azonos adatokkal számítja ki a C értékét. Igaz, a program utasításait módosíthatjuk. Pl. A-nak új értéket adhatunk, ha a kurzort a 20-as sor 3. karakterére állítjuk és új számot írunk be, pl. 5.6-ot, majd a RETURN billentyű lenyomásának hatására a módosított utasítás kerül a tárbá. Hasonlóan módosítható a 30-as sor utasítása is, ha a kurzort a 2. karakterére állítjuk és új számértéket írunk be, pl. 10-et. Ekkor a RETURN billentyű lenyomásával ez a módosított utasítás is a tárbá kerül. Egy új sorba beírva a RUN parancsot és a RETURN billentyűt lenyomva a

```
10 PRINT "2":REM*** 2. PROGRAM
14 :
15 REM** AZ 1. PROGRAM MODOSITASA
16 REM** A-NAK, B-NEK UJ ERTEKET ADUNK
17 :
20 A=5.6
30 B=10
40 C=A/B+3*8
50 PRINT "C=";C
```

program végrehajtásával a C=.0448 eredmény jelenik meg a képernyő felső sorában.

A következőkben összefoglaljuk és kiegészítjük a számológép-üzemmódban használt parancsok utasításkénti használatát és ismertetjük azok teljes körű alkalmazását.

A PRINT kulcsszó

A kiíratási utasítás alakja:

s PRINT lista,

ahol a lista szám konstansok, füzérkonstansok, bármelyik típusú változó és kifejezések sorozata lehet.

A lista egyes elemeit pontosvessző (;) vagy vessző (,), ill. a SPC(ak) vagy a TAB(ak) parancsok választhatják el. Mindegyik a képernyőkép (nyomtatási kép) kialakítására van hatással:

; — szóköz nélküli kiírást eredményez. Az utolsó listaelem utáni pontos-

vessző hatására a következő PRINT a kiírást az előző kiírás utáni első karakterhelyen folytatja.

- képernyő 80 karakteres logikai sora 8-szor 10 oszlopos kiíratási zónára van felosztva. A vessző hatására a vessző után következő érték kiírását a következő zónában kezdi. A vessző a lista utolsó eleme után is állhat.

SPC(*ak*) — az *ak* aritmetikai kifejezés, ill. konstans értékével azonos számú szóközt ír ki. Az SPC és a kezdőzárójel között nem lehet szóközt hagyni, mert az értelmező SP(*ak*) tömbelemként tekinti. Az SPC(*ak*) csak a PRINT utasítás listájában szerepelhet, önálló parancsként vagy utasításként nem. ($0 \leq ak \leq 255$, ellenkező esetben ? ILLEGAL QUANTITY ERROR IN *s* üzenetet kapunk, ahol az *s* a hibát tartalmazó utasítássor sorszáma.)

TAB(*ak*) — a képernyő bal oldalától számítva, az *ak* aritmetikai kifejezés, ill. konstans értékének megfelelő számú szóköz kihagyásával ír ki. A TAB és a kezdőzárójel között nem lehet szóköz, mert az értelmező TAB(*ak*) tömbelemként tekinti. A TAB(*ak*) csak a PRINT utasítás listájában szerepelhet, önálló parancsként vagy utasításként nem. ($0 \leq ak \leq 255$, ellenkező esetben

? ILLEGAL QUANTITY ERROR IN *s*

üzenetet kapunk, ahol az *s* a hibás utasítássor sorszáma.)

A PRINT kulcsszó a program utasításaiban is helyettesíthető kérdőjellel.

Tanulmányozzuk az előbbi listaelemek hatásait a következő program kiírásai alapján!

```
10 PRINT "A":REM*** 3. PROGRAM
14 :
15 REM** A PRINT, A VESSZO, A PONTOSVESSZO, AZ SPC
16 REM** ES A TAB HATASAINAK SZEMLELTETESE.
17 :
20 PRINT "ALMA ", "MEGGY-"; "FA:", 34, "A"
30 PRINT "ALMA" SPC(4) "MEGGY-"; "FA:"; 34 SPC(6) "A"
40 PRINT "AA" TAB(5) "BB" TAB(16) "CC"
50 PRINT "0123456789012345678901234567890123456789"
60 PRINT "VARGA ";
70 PRINT "KATA"
```

A program futtatásának képernyőképe:

```
ALMA      MEGGY-FA:  34      A
ALMA      MEGGY-FA:  34      A
AA BB      CC
0123456789012345678901234567890123456789
VARGA KATA

READY.
SYS36864
```

Vegyük figyelembe, hogy az értelmező a számkonstansok kiírásakor automatikusan egy szóközt illeszt a szám után, valamint pozitív számnál az előjel helyett szintén egy szóközt jelenít meg.

A RUN kulcsszó

A programvégrehajtás parancs, ill. utasítás alakja:

```
RUN, ill.      RUN s1
s RUN, ill. s RUN s1,
```

ahol *s* és *s1* a program utasítássorainak sorszámai. A RUN, ill. *s* RUN a tárolt programot előlről, a RUN *s1*, ill. *s* RUN *s1* a tárolt programot az *s1*-es sorszámu utasítástól kezdve hajtja végre. A program végrehajtása előtt a program változóinak előző értékeit törli.

Ha a programban nincs *s1*-es sorszámu utasítássor, akkor a program végrehajtása az

? UNDEF' D STATEMENT ERROR

üzenettel leáll.

A REM kulcsszó

Megjegyzések programba írására való parancs vagy utasítás alakja:

```
s utasítás: REM karaktersorozat
s REM karaktersorozat,
```

A REM lehetővé teszi, hogy a programban az olvasónak tájékoztató megjegyzéseket helyezzünk el. A REM utáni *karaktersorozat* tetszőleges lehet.

A LIST kulcsszó

A tárolt program listázásának parancs, ill. utasítás alakja:

```
LIST s1 - s2      s LIST s1 - s2,
```

ahol *s1* és *s2* a tárban levő program egy-egy utasítássorának sorszáma. Ha *s1* < *s2*, akkor a LIST az *s1*-es sorszámu sortól az *s2*-es sorszámu sorig írja ki a képernyőre a program sorait.

Ha nem adjuk meg az *s1*-et, akkor a

```
LIST - s2
```


a programot az első sorától az s2-es sorszámu sorig, ha nem adjuk meg az s2-t, akkor a

LIST s1 —

hatására a programot az s1-es sorszámu sorától az utolsó soráig, ha egyiket sem adjuk meg, akkor a

LIST

a teljes programot, ha pedig csak az s1-et adjuk meg, akkor a

LIST s1

az s1-es sorszámu sort írja ki a képernyőre.

Megjegyzések:

1. A tárból a képernyőre írt program utasítássorai a BASIC szabványai szerint jelennek meg. Így pl., ha a program írásakor kérdőjelet használtunk a PRINT kulcsszó helyett, akkor a listázáskor PRINT fog megjelenni; ha szóköz nélkül írtuk a sorszámot és az azt követő utasítást, a listázáskor a képernyőn szóközzel fog megjelenni stb.

2. Ha egy program valamelyik utasítássorában a LIST kulcsszóval szerepel egy utasítás, akkor annak a végrehajtása után a program futása megszakad. Ezért ilyen utasítást csak utolsó utasításként célszerű használni.

A NEW kulcsszó

Parancs, ill. utasítás alakja:

NEW s NEW

A NEW paraméter nélküli parancs, ill. utasítás. Végrehajtása törli a programot és a változókat a tárból.

Új program írásának megkezdése előtt — ha nem akarjuk, hogy az új program a tárból levővel összekeveredjék — feltétlenül adjuk ki a NEW parancsot!

Figyelem! A programban elhelyezett s NEW végrehajtja a NEW parancsot, azaz törli a programot.

1.6 BEVITELI UTASÍTÁSOK I. Az INPUT kulcsszó

A könyvben ismertetésre kerülő *beviteli utasítások* lehetővé teszik, hogy a számítógép billentyűzetéről közvetlenül adatokat (számkonstanst, füzérkonstanst) adjunk át egy működő BASIC program változóinak.

Az INPUT kulcsszó

Az INPUT beviteli utasítás alakja:

s INPUT "karaktersorozat"; lista

ahol s az utasítás sorszáma, a „karaktersorozat” pl. tájékoztató szöveget tartalmazhat a változókról, de a pontosvesszővel együtt el is hagyható az utasításból; a lista elemei vesszővel elválasztott az egyszerű vagy tömbváltozók.

Az INPUT utasítás végrehajtásakor a képernyőre kiíródik a *karaktersorozat*, majd közvetlen utána egy kérdőjel, s a kérdőjel után megjelenik a kurzor. A számítógép a listában felsorolt összes változóhoz megfelelő típusú adatot vár. A listában vesszővel elválasztott változókhoz az adatokat is vesszővel elválasztva vagy új sorba kell bebillentyűzni.

Ha a lista minden változója megfelelő értéket kapott, akkor a RETURN billentyű lenyomásával a program végrehajtása folytatódik.

Ha az INPUT után idézőjelek közé zárt karaktersorozat szerepel, akkor a befejező idézőjel után kötelező a pontosvessző (;) beírása, mert elhagyása esetén a

? SYNTAX ERROR IN s

üzenet jelenik meg a képernyőn (s a hibás utasítás sorszáma).

Ha nem kapott még értéket a lista összes eleme, de az utoljára leírt adat után nem tettünk vesszőt és lenyomtuk a RETURN billentyűt, akkor a következő sor elején két kérdőjel (??) és a kurzor jelenik meg, jelezve, hogy még adatot vár a gép. Ellenben ha vesszőt teszünk az utolsó listaelem értékadása előtt és lenyomjuk a RETURN billentyűt, akkor az utolsó elem 0 értéket kap, és a program hibajelzés nélkül fut tovább.

Ha a lista utolsó változója számára beírt adat után teszünk vesszőt, vagy több adatot adunk meg, akkor a RETURN billentyű lenyomása után

? EXTRA IGNORED

(a főleg figyelmen kívül marad) üzenetet kapunk, de a program tovább fut.

Ha numerikus változóhoz füzérkonstanst rendelünk, ha akkor

? REDO FROM START

üzenetet kapunk és ismét végrehajtódik az INPUT utasítás.

A füzérváltozó számára akár idézőjelek között, akár idézőjelek nélkül adjuk meg a karaktersorozatot, azonos módon tárolódik.

Az INPUT parancsként nem használható!

Az elmondottakat tanulmányozhatjuk a következő programmal, ha a változók-nak különböző adatokat adunk.

```
10 PRINT "M":REM*** 4.PROGRAM
11 REM** PITAGORASZ-TETEL ALKALMAZASA
12 :
13 PRINT:PRINT " IRD LE A DEREKSZOGU 3-SZOG"
14 PRINT " BEFOGOINAK HOSSZAT ES AZ ATFOGO"
15 PRINT " JELET A PROGRAM KERESE SZERINT":PRINT
17 :
20 PRINT " A HAROMSZOG ADATAI:":PRINT
22 INPUT " 1. BEFOGO Q=":Q
23 INPUT " 2. BEFOGO W=":W
24 INPUT " ATFOGO A$=":A$
30 PRINT:PRINT " Q=";Q,"W=";W,"A$=";A$
40 X=(Q^2+W^2)^(.5):REM** PITAGORASZ-TETEL ALK.
50 PRINT:PRINTTAB(10)A$;"=";X
```

A RUN **RETURN** után a képernyő utolsó szövegsorában

1. BEFOGO Q=?

jelenik meg. Írjunk be a villogó kurzor helyére pl. 3-at és nyomjuk le a **RETURN** billentyűt! Ekkor a következő sorban a

2. BEFOGO W=?

jelenik meg. Írjunk be pl. 4-et és nyomjuk meg a **RETURN** billentyűt! A képernyő következő sorában az

ATFOGÓ A\$=?

jelenik meg. Írjunk be pl. az ATFOGO karaktersorozatot és nyomjuk le a **RETURN** billentyűt! Megjelennek a program eredmény sorai.

A program futtatásának képernyőképe:

```
IRD LE A DEREKSZOGU 3-SZOG
BEFOGOINAK HOSSZAT ES AZ ATFOGO
JELET A PROGRAM KERESE SZERINT
```

A HAROMSZOG ADATAI:

```
1. BEFOGO Q=? 3
2. BEFOGO W=? 4
ATFOGO A$=? ATFOGO
```

```
Q= 3      W= 4      A$=ATFOGO
ATFOGO = 5
```

READY.
SYS36864

1.7 ARITMETIKAI FÜGGVÉNYEK

A BASIC lehetővé teszi, hogy az utasításokban néhány elemi függvény függvényértékét közvetlenül kiszámítva használhassuk.

Pl. a $\cos x$ függvényérték kiszámítását, egy radiánban megadott x értékhez, a BASIC-ben a $\text{COS}(X)$ kifejezéssel valósíthatjuk meg. A BASIC-ben használható függvények bármelyikének értékét a függvényazonosító és az azt követő zárójelpárba írt argumentum együtt adja meg, mint elsődleges aritmetikai kifejezés. A függvényazonosító és a kezdőzárójel között nem lehet szóköz.

Az elemi függvény argumentuma tetszőleges aritmetikai kifejezés lehet. A BASIC lehetővé teszi, hogy egy programban új egyváltozós függvényt definiáljunk és azt abban a programban akárhányszor felhasználjuk (l. az 1.11 alfejezetben).

BASIC függvénykifejezések

ABS(X) Az X argumentum abszolút értékét képezi:

$$\text{ABS}(X) = \begin{cases} X, & \text{ha } X \geq 0; \\ 0, & \text{ha } X = 0; \\ -X, & \text{ha } X < 0. \end{cases}$$

SIN(X) A radiánban adott X argumentum szinuszt számítja ki.

COS(X) A radiánban adott X argumentum koszinuszt számítja ki.

TAN(X) A radiánban adott X argumentum tangensét számítja ki a

$$\text{tg } x = \frac{\sin x}{\cos x} \text{ összefüggés alapján,}$$

s így $\cos x \approx 0$ esetén

? DIVISION BY ZERO ERROR

üzenetet kaphatunk.

Ez pl. $\text{TAN}(\pi/2 - 1E-10)$ esetén már bekövetkezik.

ATN(X) Az X argumentum arkusz tangensének $\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$ intervallumba eső értékét számítja ki radiánban.

EXP(X) Az e^x függvényértéket számítja ki, ahol $e \approx 2.71828183$. Ha $X > 88.0296919$, akkor
? OVERFLOW ERROR
üzenetet kapunk, ha $X < -88.0296912$, akkor 0 függvényértéket kapunk.

LOG(X) Az X argumentumnak az e alapú, azaz a természetes logaritmusát számítja ki. Ha $X \leq 0$, akkor
? ILLEGAL QUANTITY ERROR
üzenetet kapunk. Az 10-es alapú logaritmust a $\text{LOG}(X)/\text{LOG}(10)$ formulával számíthatjuk ki.

SQR(X) Az X argumentum négyzetgyökét számítja ki, ha $X \geq 0$. Ha $X < 0$, akkor
? ILLEGAL QUANTITY ERROR
üzenetet kapunk.

INT(X) Az X argumentum egész értékét képezi, amely az X-nél kisebb, de hozzá legközelebb eső egész szám.
Pl.: $\text{INT}(12.456) = 12$
 $\text{INT}(-12.456) = -13$

RND(X) Az RND a $[0; 1]$ intervallumba eső ún. pseudovéletlen lebegőpontos számokat állít elő. Az értelmező az utoljára előállított véletlen számot megőrzi és kezdőértékként felhasználja a következő véletlen szám kiszámításához.
A számítógép előállít egy kezdőértéket a bekapcsolásakor.
Ha az argumentum pozitív, akkor az RND függvényértékek, azaz a pseudovéletlen számok a bekapcsoláskor adott kezdőértékből képződnek.
Ha az argumentum negatív, akkor az RND függvény minden híváskor új kezdőértéket képez. Ha az argumentum negatív számkonstans ($X < 0$), akkor minden negatív számkonstanshoz külön-külön, de változatlan értéket állít elő.
Pl.: $\text{RND}(-1) = 2.99196472 \text{ E}-08$
 $\text{RND}(-2) = 2.99205567 \text{ E}-08$
 $\text{RND}(-5.5) = 4.10973371 \text{ E}-08$
Ha $X = 0$, akkor az RND az ún. szabadon futó hardver órából állít elő kezdőértéket. Az így előállított számsorozatok csak látszólag állnak véletlen számokból. Pl. a $[0; 19]$ intervallumba eső névleges véletlen egész számok előállítása:
PRINT INT(RND(0)*20)
A $[1; 20]$ intervallumba eső véletlen számok előállítása:
PRINT INT(RND(0)*20)+1
Tetszőleges A alsó és F felső határok közötti véletlen számok előállítása:
PRINT RND(1)*(F-A)+A

PEEK(X) A PEEK egy $[0; 255]$ intervallumba eső egész értéket előállító függvény, amely a tár X sorszámu rekeszében levő byte értékével egyenlő.

Az argumentumnak a $0 \leq X \leq 65535$ feltételt ki kell elégítenie, mert különben
? ILLEGAL QUANTITY ERROR
üzenetet kapunk.
Pl.: PRINT PEEK (53281) AND 15 eredményül a képernyő keretszínének kódját adja.

SGN(X) Egész értékű függvény, amely 1, 0 vagy -1 értéket állít elő, attól függően, hogy az X argumentum pozitív, 0 vagy negatív értékű, azaz

$$\text{SGN}(X) = \begin{cases} 1, & \text{ha } X > 0; \\ 0, & \text{ha } X = 0; \\ -1, & \text{ha } X < 0. \end{cases}$$

Írjunk egy programot a $\sin(6x) = 2 \sin(3x) \cos(3x)$ azonosság jobb és bal oldalán levő kifejezések kiszámítására, s néhány x értékre vizsgáljuk meg, hogy a kiszámított értékek azonosan egyenlők-e egymással!

NEW

```
10 PRINT "N":REM*** 5. PROGRAM
11 :
12 PRINT " ARITMETIKAI FUGGVENYERTEK "
13 PRINT " KISZAMITASANAK SZEMLELTETESE ES A "
14 PRINT " PONTOSSAG VIZSGALATA.":PRINT
15 :
20 INPUT " AZ ARGUMENTUM ERTEKE X=":X:PRINT
30 Z=2*SIN(3*X)*COS(3*X)
40 V=SIN(6*X)
50 A=V-Z:REM** V=Z RELACIO
60 PRINT " A TRIG. KIFEJEZES BAL OLDALA: "
62 PRINT " V=":V:PRINT
65 PRINT " A TRIG. KIFEJEZES JOBB OLDALA: "
66 PRINT " Z=":Z
70 PRINT:PRINT " A=":A:REM** A A RELACIO IGAZ VAGY
72 REM** HAMIS ERTEKET ADJA MEG
```

Ha a programot egymás után pl. $x=0$, $x=.5$, $x=2$ értékekkel futtatjuk, akkor Z-re és V-re mindhárom esetben azonos értékeket kapunk, a reláció értéke $A=0$, de pl. $X=6$ -ra, bár a képernyőre kiírt Z és V azonos, azonban $A \neq 0$, tehát a kiírás előtti kerekítetlen Z és V értékek nem egyenlők. Ha $X=\pi/3 \approx 1.04719755$ értékkel futtatjuk a programot, akkor

$Z = -5.85167231 \text{ E}-09$
 $C = -5.85167232 \text{ E}-09$
 $A = 0$

értékeket kapjuk, vagyis a kerekítési hiba a nyolcadik tizedesjegyben jelentkezik.

1.8 VEZÉRLÉSÁTADÓ UTASÍTÁSOK I.

A GOTO, az IF...THEN..., a STOP, az END és a CONT kulcsszavak

A számítógép a program utasításait mindaddig az utasítássorszámok növekvő értékének megfelelő sorrendben, egymás után hajtja végre, amíg ezt a feldolgozási módot egy utasítás meg nem változtatja. A sorrend megváltoztatását, ill. az új sorrend kijelölését a *vezérlésátadó utasításokkal* lehet megvalósítani.

A GOTO kulcsszó

A feltétlen vezérlésátadást a GOTO kulcsszóval írjuk elő, amelynek parancs, ill. utasítás alakja:

GOTO *s1* *s* GOTO *s1*

ahol *s1* annak az utasításnak a sorszáma, amelyikkel folytatni kívánjuk a programot. A GOTO szóközzel is írható: GO TO. A GOTO után, ha nem írunk sorszámot, akkor a GOTO 0 kerül végrehajtásra, s mivel 0 sorszámot általában nem adunk,

? UNDEF'D STATEMENT ERROR

üzenetet kapunk. Ezt az üzenetet kapjuk minden esetben, ha olyan sorszámot adunk meg, amilyen a programban nem fordul elő.

A GOTO-val könnyen szervezhetünk végtelen ciklust, pl. a

100 GOTO 100

utasítás végtelen ciklust hoz létre, amelyből csak a

RUN
STOP

billentyű lenyomásával léphetünk ki. Hatására a program végrehajtása megszakad és a

BREAK IN 100

üzenet jelenik meg a képernyőn.

Az IF...THEN... kulcsszavak

Feltételes vezérlésátadást az

s IF *k* THEN *s1*
s IF *k* GOTO *s1*
s IF *k* THEN *parancsok*

utasítások bármelyikével megvalósíthatunk, ahol *s*, *s1* utasítássorszám, *k* tetszőleges BASIC kifejezés. Ha a *k* kifejezés igaz, vagyis az általa előírt feltétel teljesül, akkor az első és a második alakban felírt utasítás az *s1*-es sorszámú utasításra adja át a vezérlést, a harmadik alakban felírt utasítás pedig a THEN után álló *parancsokat* hajtja végre.

Ha a *k* kifejezés hamis, vagyis az általa előírt feltétel nem teljesül, akkor az utasítás a következő programsorra adja át a vezérlést.

A második alakú feltételes vezérlésátadó utasításban GOTO nem írható szóközzel (GO TO), mert

? SYNTAX ERROR IN *s*

üzenetet kapunk.

Ha *k* aritmetikai kifejezés, akkor csak a 0 értéke számít hamisnak. Pl.: a

10 IF X+2+1 THEN 10

utasítás X bármely értékére a 10 GO TO 10 utasítással azonos hatású.

Pl.: NEW

```
10 PRINT "REM*** 6. PROGRAM
12 :
14 REM** A 80-AS SORSZAMU UTASITAS VEGTELEN CIKLUST
15 REM** HOZ LE'TRE
16 :
20 W=1
30 A=1
40 PRINTW,A:X=X-1:W=W+1:A=A+1
50 IF X+2 GOTO 40
60 PRINT:PRINT " VEGTELEN CIKLUS KOVETKEZIK!"
70 PRINT " A RUN/STOP BILLENTYUT KELL LEUTNI!"
80 IF X+2+1 THEN 80 :REM** X+2+1 > 0
```

Az 50-es sorszámú utasítás, amíg $X+2 > 0$, a 40-es sorszámúra adja a vezérlést, amikor $X+2 = 0$ lesz, akkor a 60-as sorszámú utasítással folytatódik a program, majd a 80-as sorszámú utasítás végtelen ciklust hoz létre, mivel $X+2+1$ aritmetikai kifejezés minden X értékre pozitív marad.

Írjunk programot az

$$ax^2 + bx + c = 0$$

másodfokú egyenlet valós gyökeinek kiszámítására az

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

formulát alkalmazva!


```

10 PRINT "Q":REM*** 7.PROGRAM
12 :
13 PRINTTAB(10)" AZ A*X+2+B*X+C=0 ":PRINT
20 PRINT " MASODFOKU EGYELET VALOS GYOKEI":PRINT
22 :
30 PRINT " KEREM AZ A,B,C EGYUTTHATOKAT"
32 INPUT " A=";A:INPUT " B=";B:INPUT " C=";C
40 IF A=0 THEN PRINT:PRINT " NEM MASODFOKU.":STOP
50 D=B+2-4*A*C
60 IF D<0 THEN PRINT:PRINT " NINCS VALOS GYOK.":STOP
70 A2=2*A
80 D1=SQR(D)
90 X1=(-B+D1)/A2:X2=(-B-D1)/A2
100 PRINT:PRINTTAB(10)" VALOS GYOKOK:":PRINT
110 PRINTTAB(3)" X1=";X1SPC(4)" X2=";X2
120 END

```

A STOP és az END kulcsszavak

Az előbbi programban két új BASIC kulcsszó is előfordul, a STOP és az END.

A STOP megszakítja a program futását és kiírja azt, hogy hányadik sorban állította meg a programot

BREAK IN s

üzenet formájában. Hatása megegyezik a RUN/STOP billentyű lenyomásának hatásával.

Az END, a program vége utasítás, a rendszernek adja át a vezérlést.
Pl.:

```

10 PRINT "Q":REM*** 8.PROGRAM
11 :
12 PRINT:PRINT " IRJ HAROM SZAMOT VESSZOVEK "
13 PRINT " ELVALASZTVA. ":PRINT
14 PRINT " HA A SZAMOK EGYENLOK EGYMASSAL, AKKOR"
15 PRINT " A 30-AS UTASITASNAL, HA NEM EGYENLOK,"
16 PRINT " AKKOR A 40-ES UTASITASNAL ALL LE"
17 PRINT " A PROGRAM.":PRINT
18 :
20 INPUT Q,W,E:PRINT
30 IF Q=W AND W=E THEN PRINT " Q=W=E, STOP.":STOP
40 STOP

```

Ha a Q, W, E változók azonos értéket vesznek fel, pl. Q=1, W=1, E=1, akkor

BREAK IN 30

üzenettel áll le a program.

Ha a három változó értéke közül bármelyik különbözik a másik két változó értékétől, akkor a

BREAK IN 40

üzenettel áll le a program.

Az END leállítja a program futását és a vezérlést visszaadja a programszerkesztőnek, de a

BREAK IN s

üzenet helyett a képernyőn csak a READY. felirat jelenik meg.

Pl.:

```

10 PRINT "Q":REM*** 9.PROGRAM
20 PRINT " INDITHATO A PROGRAM?( IGEN/NEM) ":PRINT
26 PRINT " A NEM LEIRASA ESETEN A 40-ES UTASITAS"
27 PRINT " LEALLITJA A PROGRAMOT, EGYEBKENT"
28 PRINT " PEDIG AZ 50-ES UTASITASSAL FOLYTATODIK."
30 INPUT Q$
40 IF Q$="NEM" THEN END
50 PRINT:PRINT " HA VALASZUNK ELTER A NEM- TOL,"
52 PRINT " A PROGRAM FOLYTATODIK."
60 END

```

Ha Q\$="NEM", akkor a 40-es utasítással END parancsára, ha Q\$<>"NEM", akkor pedig az 50-es és 52-es utasítással végrehajtása után a 60 END utasítás hatására áll le a program.

A STOP és az END utasításokkal leállított program a C 64-es számítógépen a

CONT

paranccsal folytatható. Így pl., ha a 9. program Q\$="NEM" hatására leállt, akkor a CONT **RETURN** után végrehajtnak a 40-es sort követő utasítások is. A változók értéke változatlanul megmarad, így pl. a

PRINT Q\$ **RETURN**

parancs hatására a NEM szót írja ki.

A 8. programban Q=W=E esetén a 30-as utasításban levő STOP hatása érvényesül. A változók értékét PRINT Q,W,E parancs végrehajtásával kiirathatjuk a képernyőre, majd a CONT paranccsal a program végrehajtását folytathatjuk.

Példák:

1) Írjunk programot egy A szám négyzetgyökének kiszámítására a

$$\sqrt{A} \approx \frac{1}{2} \left(x_{n-1} + \frac{A}{x_{n-1}} \right) = x_n, \quad (n = 1, 2, \dots)$$

iterációs algoritmus felhasználásával, az

$$|x_n - x_{n-1}| \leq H$$

hibakorlát teljesülése mellett, ahol legyen

$$H=1E-8.$$

A gyök kiszámítása után írassuk ki az A , az iterációval kapott közelítőgyök és az $SQR(A)$ értékét!

```
10 PRINT "□":REM*** 10.PROGRAM
11 :
20 PRINT " ITERACIOS ALGORITMUS NEGYZETGYOK "
22 PRINT " KISZAMITASARA.":PRINT
23 :
30 INPUT " HIBAKORLAT=":H
40 PRINT:INPUT " SZAM A=":A
50 X=A/2:REM** A KEZDO ERTEK 0. KOZELITESE
60 B=.5*(X+A/X)
70 IF ABS(X-B)<=H THEN 100
80 X=B
90 GOTO 50
100 PRINT:PRINT:PRINT " A=":A:PRINT
110 PRINT " IT. GYOK=":B:PRINT " SQR(A)=":SQR(A)
120 PRINT:PRINT " BEFEJEZTE A SZAMOK GYOKENEK "
122 PRINT " MEGHATAROZASAT?( IGEN/NEM)":PRINT
130 INPUT V$
140 IF V$="NEM" THEN 40
150 PRINT:PRINTTAB(16)"V E G E ."
160 END
```

2) Írjunk programot a harmadfokú egyenlet megoldására a *Cardano*-képlet alapján.

A harmadfokú egyenlet:

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0.$$

A számítás menete:

$$Q = \frac{3a_1 - a_2^2/a_3}{9a_3}$$

$$R = \frac{9a_2a_1/a_3 - 27a_0 - 2a_2^3/a_3^2}{54a_3}$$

Ha $Q^3 + R^2 < 0$, akkor az egyik valós gyök:

$$x_3 = 2\sqrt{-Q} \cos\left(\frac{1}{3} \arccos\left(R/\sqrt{-Q^3}\right)\right) - \frac{a_2}{3a_3}.$$

Megjegyzés; az $\arccos$ függvényértéket az

$$\arccos(x) = -\arccos(x/\sqrt{1-x^2}) + \frac{\pi}{2}$$

összefüggés alapján számítjuk ki.

Ha

$$Q^3 + R^2 \geq 0, \text{ akkor}$$

$$S = \sqrt[3]{R + \sqrt{Q^3 + R^2}}$$

$$T = \sqrt[3]{R - \sqrt{Q^3 + R^2}}$$

és az egyik valós gyök:

$$x_3 = S + T - \frac{a_2}{3a_3}$$

Az x_3 kiszámítása után másodfokúra redukáljuk az egyenletet:

$$b_2x^2 + b_1x + b_0 = 0$$

választás esetén:

$$b_2 = 1.00$$

$$b_1 = x_3 + \frac{a_2}{a_3}$$

$$b_0 = \left(x_3 + \frac{a_2}{a_3}\right)x_3 + \frac{a_1}{a_3}$$

Ha $b_1^2 - 4b_2b_0 \geq 0$, akkor a valós gyökök:

$$x_1 = \begin{cases} (-b_1 - \sqrt{b_1^2 - 4b_2b_0})/(2b_2), & \text{ha } -b_1/b_2 < 0 \\ (-b_1 + \sqrt{b_1^2 - 4b_2b_0})/(2b_2), & \text{ha } -b_1/b_2 \geq 0 \end{cases}$$

és

$$x_2 = \frac{b_0}{b_2x_1}.$$

Ha $b_1^2 - 4b_2b_0 < 0$, akkor a komplex gyökök:

$$x_{1,2} = -\frac{b_1}{2b_2} \pm i \frac{\sqrt{|b_1^2 - 4b_2b_0|}}{2b_2}$$

10 PRINT "□":REM*** 11.PROGRAM ***

```
11 :
12 :
20 PRINT:PRINT " HARMADFOKU EGYENLET MEGOLDASA"
22 PRINT:PRINT
30 PRINT " A HARMADFOKU EGYENLET EGYUTTHATOIT"
32 PRINT " A3,A2,A1,A0 SORRENDENBEN KEREM!"
34 :
40 INPUT " A3=":A3
42 INPUT " A2=":A2
44 INPUT " A1=":A1
46 INPUT " A0=":A0:PRINT
```



```

50 IF A3<>0 THEN 60
52 PRINT " A3=0, EZ NEM HARMADFOKU EGYENLET!":GOTO 30
53 :
54 REM *****
55 REM AZ EGYIK VALOS GYOK KISZAMITASA:
56 REM *****
57 :
60 Q=(3*A1-A2↑2/A3)/(9*A3)
70 R=(9*A2*A1/A3-27*A0-2*(A2↑3/A3↑2))/(54*A3)
80 D=Q*Q+R*R
82 IF D>=0 THEN 100
84 Z=R/SQR(-Q↑3)
85 X3=2*SQR(-Q)*COS(1/3*(-ATN Z/SQR(1-Z↑2))+π/2))
86 X3=X3-A2/(3*A3)
94 GOTO 140
100 D1=SQR(D)
110 IF (R+D1)<0 THEN S=-ABS(R+D1)↑(1/3):GOTO 116
112 S=(R+D1)↑(1/3)
116 IF (R-D1)<0 THEN T=-ABS(R-D1)↑(1/3):GOTO 130
120 T=(R-D1)↑(1/3)
130 X3=S+T-A2/(3*A3)
132 :
134 REM *****
135 REM A MASODFOKU EGYENLET MEGOLDASA:
136 REM *****
137 :
140 B2=1:B1=X3+A2/A3
150 B0=(X3+A2/A3)*X3+A1/A3
160 D2=B1↑2-(4*B2*B0)
170 D3=SQR(ABS(D2)):H=2*B2
180 IF D2<0 THEN 220
190 IF -B1/B2<0 THEN X1=(-B1-D3)/H:GOTO 210
200 X1=(-B1+D3)/H
210 X2=B0/(B2*X1):GOTO 300
220 X2=-B1/H:X2=X2/H
240 PRINT "□":PRINT:PRINT " KET GYOK KOMPLEX:"
250 PRINT:PRINT " X1=";X1;"+";"X2;"*I"
260 PRINT " X2=";X2;"-(";"X2;"*I"
262 IF ABS(X1)<1E-3 THEN PRINT:GOTO 264
263 GOTO 270
264 PRINT:PRINT " MEGJEGYZES!":PRINT
265 PRINT " MIVEL ABS(IM(X1))<1E-3, A GYOKOK"
266 PRINT " VALOSAK IS LEHETNEK."
270 PRINT:PRINT " EGY GYOK VALOS:"
280 PRINT:PRINT " X3=";X3
290 PRINT:PRINT:GOTO 340
300 PRINT "□":PRINT:PRINT:PRINT " MINDHAROM GYOK VALOS:"
310 PRINT:PRINT " X1=";X1
320 PRINT " X2=";X2
330 PRINT " X3=";X3
340 PRINT " AKAR UJABB EGYENLETET MEGOLDANI? (I/N)"
350 INPUT G$:IF G$="I" THEN PRINT:GOTO 30
360 PRINT:PRINT TAB(10) " BEFEJEZVE."
370 END

```

1.9 CIKLUSSZERVEZŐ UTASÍTÁSOK A FOR...TO...STEP... NEXT kulcsszavak

Az IF ... THEN ... feltételes vezérlésadó utasítással is tudunk olyan programot írni, amelynek egy részét pl. egy feltétel teljesüléséig függően többször megismétel-tetünk.

Ciklusnak nevezzük az utasítások olyan véges sorozatát, amelyek ismételten végrehajtandók, mindaddig, amíg egy meghatározott feltétel fennáll.

Pl.:

```

10 PRINT "□":REM*** 12.PROGRAM
12 :
14 REM** AZ IF...THEN VIZSGALATA
15 :
20 PRINT:PRINT " NEGYZETGYOK ERTEKEK 1-5-IG:"PRINT
22 :
24 :
30 I=1
40 PRINT " SQR(";"I";")=";SQR(I)
50 IF I<5 THEN I=I+1:GOTO 40
60 PRINT:PRINT " AZ I ERTEKET A VIZSGALAT UTAN"
62 PRINT " NOVELTUK."
70 PRINT:PRINT
100 I=1
110 PRINT " SQR(";"I";")=";SQR(I)
120 I=I+1
130 IF I<5 THEN 110
140 PRINT:PRINT " AZ I ERTEKET A VIZSGALAT ELOTT"
142 PRINT " NOVELTUK."
150 END

```

Tanulmányozzuk át a 30—60-as és a 100—140-es utasítássorokat! A 40-es utasítás-sor az I=1, 2, 3, 4, 5 értékekre, a 110-es utasítássor az I=1, 2, 3, 4 értékekre kerül végrehajtásra; annak ellenére, hogy az 50-es és a 130-as utasítássor is az I<5 feltételt vizsgálja. A különböző eredmény abból adódik, hogy az első esetben a vizsgálat után növeljük I értékét, a második esetben pedig a vizsgálat előtt.

A BASIC az ilyen típusú ciklusok szervezésének megkönnyítésére a

```
FOR V= K1 TO K2 STEP L
```

```
.
```

```
.
```

```
.
```

```
NEXT V
```

utasítások használatát teszi lehetővé, ahol V a ciklusváltozó, K1 aritmetikai kifejezés a ciklusváltozó kezdőértéke, K2 aritmetikai kifejezés a ciklusváltozó felső határa, L aritmetikai kifejezés pedig a ciklusváltozó növekménye, amellyel a NEXT V utasításig terjedő programrész minden ismétlése után megnövekszik a ciklusváltozó értéke (lépésköz).

A FOR ... és NEXT utasítások közötti részt *ciklusmagnak* nevezzük.

A FOR ... NEXT utasításokkal írt program az IF ... THEN ... utasításokkal leírva a következőképpen működik:

```

      10 V=K1
      20 ...
      30 ...
      :
      100 V=K1+L
      110 IF V<=K2 THEN 20
      120 ...      :REM***HA V>K2

```

ha $V \leq K2$

Amint látjuk, a ciklusváltozó értéke a feltétel vizsgálata előtt növekszik az L növekménnyel (lépésközzel).

A ciklusszervező utasítás STEP L része elhagyható, ha $L=1$; ui. az értelmező ilyen esetben L értékét 1-nek veszi.

Írjuk át a 12. programot a FOR ... NEXT utasításpár használatával!

```

10 PRINT "12":REM*** 13.PROGRAM
12 :
20 PRINT:PRINT"FOR...TO...STEP..."
30 PRINT"....."
40 PRINT"....."
50 PRINT"NEXT..."
60 PRINT:PRINT" CIKLUSSZERVEZES.":PRINT
62 :
70 FOR I=1 TO 5 STEP 1
80 PRINT"SQR(1);"="SQR(1)
90 NEXT I
100 PRINT" A CIKLUS MAGJA, A 260-AS SORSZAMU"
110 PRINT" UTASITAS 5-SZOR FUTOTT LE.":PRINT
120 FOR I=1 TO 4
130 PRINT"SQR(1);"="SQR(1)
140 NEXT I
150 PRINT" A CIKLUS MAGJA, A 300-AS SORSZAMU"
160 PRINT" UTASITAS 4-SZER FUTOTT LE."
170 END

```

A programot lefutattva láthatjuk, hogy $K1=1$, $K2=5$ és $L=1$ esetén ötször ismétlődött, míg $K1=1$ és $K2=4$ ($L=1$) esetén négyszer ismétlődött a ciklusmag.

Figyelem! Ha a ciklusváltozó felső határa $K2$, nem egyenlő a

$K1, K1+L, K1+2L, K1+3L, \dots$

sorozat valamelyik elemével, akkor az utolsó V érték, amire végrehajtódik a ciklusmag, a $V \leq K2$ feltételnek tesz eleget.

Vizsgáljuk meg, mely értékeket írja ki a program az 1, 2, 3, 4, 5, 6 számok közül, ha $L=1, L=2, L=3$!

```

10 PRINT "13":REM*** 14.PROGRAM
20 :
30 REM** STEP L VIZSGALATA
40 :
50 PRINT:PRINT" SZAMOK NYOMTATASA 1-6 -IG "
60 PRINT" L=1,2,3 LEPESTAVOLTSAGGAL.":PRINT
70 :
80 :
90 L=1
100 FOR I=1 TO 6 STEP L
110 PRINT I;
120 NEXT I
130 PRINTSPC(6)"L=";L
140 PRINT:PRINT
150 IF L<3 THEN L=L+1:GOTO 100
160 END

```

A program futtatása utáni képernyőkép:

```

SZAMOK NYOMTATASA 1-6 -IG
L=1,2,3 LEPESTAVOLTSAGGAL.

```

```

1 2 3 4 5 6      L= 1

```

```

1 3 5      L= 2

```

```

1 4      L= 3

```

A ciklusváltozó értéke a ciklus befejezése után az aktuális $K2+L$ értékével lesz egyenlő. Így pl. a 14. program lefutása után, a PRINT I parancs végrehajtásának az eredménye: 7 ($6 < 4 + 3 = 7$).

Írjunk programot az $y = \sin(x)$ görbe és az x-tengely közötti terület közelítő kiszámítására különböző dx lépésközzel a $[0; \pi]$ intervallumban, a

$$T = \sum_{x=0}^{\pi} dx \cdot \sin(x)$$

képlet felhasználásával, amely a dx alapú és $\sin(x)$ magasságú téglalapok összegzésével közelíti az eredményt.

```

10 PRINT "14":REM*** 15.PROGRAM
12 :
20 PRINT:PRINT" SINKX> GORBE ES AZ X-TENGELY"
22 PRINT" KOZOTTI TERULET KOZELITO ERTEKE"
30 PRINT" [0;pi] INTERVALLUMBAN.":PRINT
32 :
35 T=0:DX=0.1
40 FOR X=0 TO pi STEP DX
50 T1=DX*SINKX)
60 T=T+T1
70 NEXT X
80 PRINT:PRINT" TI 0;pi=";TSPC(2)"LEPESKOZ:";DX
82 PRINT
90 PRINT" AKARJA MODOSITANI A LEPESKOZT?(I/N)"
100 PRINT: INPUT Q$
110 IF Q$="I" THEN INPUT"DX=";DX:T=0:GOTO 40
120 PRINT:PRINTTAB(12)"BEFEJEZVE.":PRINT"15"
130 END

```


A program futtatása után láthatjuk, hogy a terület mérőszámának ($T=2$) pontos értékét $5 \cdot 10^{-4}$ -nél kisebb hibával közelíti meg a programmal kiszámított érték.

Megjegyzések:

1. A ciklusváltozó nem lehet egész típusú változó, tömbváltozó, függvényváltozó, mert ilyen esetben az értelmező hibaüzenettel megszakítja a program futását. Ha a ciklusváltozó $V\%$, ill. $V(I)$ típusú, akkor

? SYNTAX ERROR IN s ,
ha $V\%$ típusú, akkor

? TYPE MISMATCH ERROR IN s

üzenetet kapjuk, ahol az s a hibát tartalmazó utasítás sorszáma.

2. Ha a FOR ... TO ... utasítás szintaktikailag helyes, akkor a ciklusmag a ciklusváltozó kezdőértékére egyszer lefut akkor is, ha a programban NEXT utasítás nem fordul elő. Az értelmező nem ad hibajelzést.

3. Ha $K2 \leq K1$ és $L > 0$, akkor hibajelzés nélkül csak a ciklusváltozó kezdőértékére, azaz egyszer fut le a ciklusmag és a program folytatódik.

4. Ha $K2 \leq K1$ és $L < 0$, akkor a ciklus szabályosan fut le.

5. Ha $L = 0$, akkor végtelen ciklus keletkezik.

6. Ha a NEXT után a ciklusváltozótól eltérő változót írunk, akkor a ciklusmag egyszer lefut a kezdőértékre és azután

? NEXT WITHOUT FOR ERROR IN s

üzenettel áll le a program.

7. Legfeljebb 9 ciklust lehet egymásba ágyazni (egymásba skatulyázni), de minden FOR-hoz tartozzék egy cikluszáró NEXT, mert ellenkező esetben lefut ugyan a program hibajelzés nélkül, de nem a kívánt eredményhez jutunk. Ez a C 64-es értelmezőjének különösen nagy hiányossága.

Pl. a 16. program az egymásba ágyazott kettős ciklust hibátlanul hajtja végre. Minden W értékhez E két értékét rendeli hozzá.

```
10 PRINT "A": REM *** 16. PROGRAM
11 :
12 PRINT " KET EGYMASBA AGYAZOTT CIKLUS "
13 PRINT "      VIZSGALATA. ": PRINT
14 :
15 PRINT "KULSO- BELSO CIKLUSVALTOZO"
20 FOR W=3 TO 1 STEP -1:PRINT
30 PRINT "===== "
40 PRINT " W="; W;
50 FOR E=1 TO 2
60 PRINT "      E="; E;
70 NEXT E
80 NEXT W
90 END
```

A program hibátlanul fut le akkor is, ha a 80-as sorszámu utasítást töröljük és a 70-es sorszámu utasítást NEXT E, W utasításra változtatjuk. (A program valamely sorának törlését a kérdéses sorszám beírása és a **RETURN** billentyű leütése végzi.)

A program akkor is helyesen fut le, ha (a 60-as és a 90-es sorok között) kétszer szerepel a NEXT utasítás a ciklusváltozókra való hivatkozás nélkül.

A program hibásan fut le, ha megcseréljük a ciklusváltozók sorrendjét a NEXT után. Pl., ha a kettős ciklust NEXT W,E utasítással zárjuk, akkor az E a második értékét nem veszi fel, a többi utasítás változatlanul végrehajtódik, majd a végén

? NEXT WITHOUT FOR ERROR IN 70

üzenetet kapjuk.

Ha csak NEXT W áll a 70-es utasítássorban és hiányzik a 80-as utasítássor, akkor a hibás programfutást nem jelzi hibaüzenettel az értelmező, de a kiírás az előzővel azonos, azaz E nem veszi fel a második értékét.

Ha csak NEXT E zárja a kettős ciklust, akkor csak a W első értékéhez rendeli hozzá E két értékét, de hibajelzést nem ad az értelmező.

Ha mindkét NEXT hiányzik a programból, akkor W első értékéhez E első értékét rendeli és hibajelzés nélkül fut tovább a program. Az ilyen hibajelzés nélküli programfutások súlyos következménnyel járhatnak!

1.10 TÖMBÖK DEKLARÁLÁSA A DIM kulcsszó

A ciklusszervező utasítással a tömbök kezelését is kényelmesebben végezhetjük el.

Ha az egy-, két- és háromdimenziós tömb bármely indexértéke 10-nél nagyobb értéket vehet fel, akkor a tömböt deklarálni kell. Háromnál több dimenziós tömböt mindig deklarálnunk kell.

A DIM kulcsszó

A tömb deklarációját a

s DIM lista

utasítással valósítjuk meg, ahol a lista elemei vesszővel elválasztva azok az egy-, két- és több dimenziós tömbök, amelyeknek a tárban elegendő helyet akarunk biztosítani a program futása során. A tömb azonosítója, típusa tetszőleges lehet.

Pl.: DIM Q(30), B%(15,15), RK\$(20), A(7), T(1,1,1,1).

A Q egydimenziós valós tömbváltozó (vektor), az index értéke 0–30 lehet. A B% kétdimenziós egész típusú tömbváltozó (mátrix), mindkét indexérték 0–15 lehet, RK egydimenziós függvény tömbváltozó, az index értéke 0–20 lehet, az A tömbváltozó számára csak a 0–7 indexértékek fordulhatnak elő, a T négydimenziós tömb pedig csak 0 és 1 indexértékű lehet.

Az egyes indexek értéke 0-tól a DIM utasításban megadott felső határig, de legfeljebb 32767-ig terjedhet. Ha ezt meghaladja valamelyik index, akkor az értelmező.

? ILLEGAL QUANTITY ERROR IN s

üzenetet ad és megszakítja a program futását.

Ha a tömb elhelyezésére nincs elegendő hely a tárban, akkor a

? OUT OF MEMORY ERROR IN s

üzenetet kapjuk.

A DIM utasításban egy tömbváltozó egy programon belül csak egyszer szerepelhet, mert máskülönben a

? REDIM'D ARRAY ERROR IN s

üzenetet kapjuk.

Ez a hibaüzenet érkezik akkor is, ha pl. a DIM listájában szereplő tömb már a DIM utasítás előtt egy értékadó utasításban, vagy más utasításban előfordult, azaz sikeres helyfoglalás történt a számára. Ezért a legcélszerűbb a tömböket a program elején deklarálni.

Ha a programban valamelyik tömbváltozó a deklarált legnagyobb indexértékénél nagyobb indexszel is előfordul, akkor a

? BAD SUBSCRIPT ERROR IN s

üzenetet kapjuk és a program futása megszakad.

A BASIC a *tömbök dinamikus deklarálását* is megengedi, így a legnagyobb indexértéket számkonstans helyett aritmetikai kifejezés is megadhatja. Pl.:

```
10 INPUT N,M
20 DIM MA(2*N+1,N*M-1)
```

Ha N=5, M=3, akkor a program az MA(11,14) mátrixtömb elemeinek biztosít helyet.

Írjunk programot két mátrix szorzatának előállítására:

$$\begin{matrix} A & \cdot & B & = & C \\ (L, M) & (M, N) & (L, N) \end{matrix}$$

ahol:

$$c_{ij} = \sum_{k=1}^L \left(\sum_{j=1}^N \left(\sum_{k=1}^M a_{ik} \cdot b_{kj} \right) \right).$$

A program a mátrixok szorzását, három egymásba ágyazott ciklussal végzi el.

```
10 PRINT"Q": REM*** 17.PROGRAM
20 :
30 :
40 PRINT:PRINT" A(L,M), B(M,N) TIPUSU TEGLALAP"
50 PRINT"      MATRIXOK SZORZASA."
60 :
70 REM** TOMBOK DINAMIKUS DEKLARALASA
80 :
90 PRINT:PRINT" L, M, N ERTEKE:" :PRINT
100 INPUT " AZ A MATRIX SORSZAMA   L=":L
110 INPUT " AZ A MATRIX OSZLOPSZAMA M=":M
120 INPUT " A B MATRIX OSZLOPSZAMA N=":N
130 DIM A(L,M),B(M,N),C(L,N):PRINT
132 :
140 PRINT" AZ A MATRIX ELEMEI OSZLOPONKENT:"
150 FOR J=1 TO M
160 FOR I=1 TO L
170 INPUT A(I,J)
180 NEXT I:PRINT:NEXT J
182 :
190 PRINT:PRINT" A B MATRIX ELEMEI OSZLOPONKENT:"
200 FOR J=1 TO N:FOR I=1 TO M
202 INPUT B(I,J)
210 NEXT I:PRINT:NEXT J
220 FOR I=1 TO L:REM*** A HARMAS CIKLUS KEZDETE
230 FOR J=1 TO N
240 S=0.0
250 FOR K=1 TO M
260 S=S+A(I,K)*B(K,J)
270 NEXT K
280 C(I,J)=S
290 NEXT J,I:REM*** A HARMAS CIKLUS VEGE
300 PRINT"Q":PRINT" A C(I,J) EREDMENY MATRIX "
310 PRINT TAB(4);L;"X";N;" TIPUSU."
320 PRINT:PRINT" ELEMEI SORONKENT:" :PRINT
330 FOR I=1 TO L
340 FOR J=1 TO N
350 PRINTSPC(2);C(I,J);
360 NEXT J
370 PRINT
380 NEXT I
390 END
```


1.11 FÜGGVÉNYEK DEFINIÁLÁSA

A DEF FN kulcsszó

Az 1.7 alfejezetben a BASIC-ben közvetlenül hívható aritmetikai függvényeket ismertettük. Gyakran, az ismertett aritmetikai függvények mellett, egyéb függvények, ill. függvénykifejezések is jól használhatók lennének számos műszaki, gazdasági feladat megoldásához. A BASIC lehetőséget ad a programozónak, hogy egy adott programon belül újabb aritmetikai függvényeket definiáljon, s azokat az aritmetikai függvényekkel azonos módon felhasználhassa a definíciót tartalmazó programban.

A függvényt definiáló utasítás alakja:

s DEF FNF(X)=ak

A DEF FN minden függvényt definiáló utasítás kulcsszója, az F a definiált függvény azonosítója, az X a függvény változója, az ak a függvényt leíró aritmetikai kifejezés.

A függvény az FNF(Z) hívó utasítással aktivizálható, amelynek hatására az X változó helyére a Z változó aktuális értéke kerül. Pl.:

```
10 DEF FN TE(R)=R*R*π
```

```
.
```

```
.
```

```
50 T=FNTE(5)
```

```
.
```

```
.
```

```
70 T1=FNTE(8)
```

```
.
```

```
.
```

A 10-es sorszámu utasítás a TE nevű függvényt definiálja, amely az R sugarú kör területét számítja ki. Az 50-es sorszámu utasítás az 5 sugarú, a 70-es sorszámu utasítás a 8 sugarú kör területét számítja ki.

A program valamely utasításában a függvény csak akkor hívható, ha azt már egy előző utasításban definiáltuk, mert ellenkező esetben az értelmező a

? UNDEF'D FUNCTION ERROR IN s

üzenetet adja és a program futását megszakítja.

Ha a függvényt definiáló utasításban az egyenlőségjel előtt hibát követünk el, akkor a

? SYNTAX ERROR IN s

üzenettel áll le a program, ahol s a definiáló utasítás sorszáma. Ha pedig az egyenlőségjel után követünk el hibát, akkor a hibaüzenet az első hívó utasítás sorszámaival jelenik meg.

A függvényt definiáló aritmetikai kifejezés bármely „beépített” aritmetikai függvényt és a programban már definiált függvényt is tartalmazhatja.

Elemizzük a következő programot!

```
10 PRINT"U": REM*** 18.PROGRAM
12 :
20 PRINT:PRINT" FUGGVENY DEFINIALASA.":PRINT:PRINT
22 :
30 DEFFN TE(R)=R*R*π:REM** KOR TERULET DEFINICIOJA
40 T=FN TE(5) :REM*** HIVAS:R=5 SUGARU KOR TER.
50 T1=FN TE(8)*SIN(1)+COS(1)*SIN(1)
55 T2=TAN(1)/2+FN TE(2):REM** FUGGVENYMUVELETEK
56 T1=T1+T2
57 :
60 PRINT:PRINT" A HIVOTT FUGGVENYEKKEL KEPZETT "
62 PRINT" EREDMENYEK.":PRINT
70 PRINT" T=";T;" T1=";T1
71 :
72 REM*** UJABB FUGGVENY DEFINICIO:
80 DEFFN NE(Z)=Z+2+Z*SIN(2*Z)*COS(1+Z+2)+FNTE(Z)
90 ER=FN NE(1) : REM*** HIVAS
100 PRINT" ER=";ER
110 END
READY.
```

Néhány matematikai függvényt definiáló utasítás:

$\arcsin(x) = \arctg\left(\frac{x}{\sqrt{1-x^2}}\right)$: DEF FN AS(X)=ATN(X/SQR(1-x²))

$\arccos(x) = -\arctg\left(\frac{x}{\sqrt{1-x^2}}\right) + \frac{\pi}{2}$: DEF FN AC(X)=-ATN(X/SQR(1-x²))+
+π/2

$\arctg(x) = \arctg(x) + \frac{\pi}{2}$: DEF FN AG(X)=ATN(X)+π/2

$\operatorname{sh}(x) = \frac{e^x - e^{-x}}{2}$: DEF FN SH(X)=(EXP(X)-EXP(-X))/2

$\operatorname{ch}(x) = \frac{e^x + e^{-x}}{2}$: DEF FN CH(X)=(EXP(X)+EXP(-X))/2

$\operatorname{th}(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$: DEF FN TH(X)=(EXP(2*X)-1)/(EXP(2*X)+1) e^{2x}+1

ar $\text{sh}(x) = \ln(x + \sqrt{x^2 + 1})$: DEF FN AS(X) = LOG(X + SQR(X² + 1))

ar $\text{ch}(x) = \ln(x + \sqrt{x^2 - 1})$: DEF FN AH(X) = LOG(X + SQR(X² - 1))

ar $\text{th}(x) = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right)$: DEF FN AT(X) = LOG((1+x)/(1-x))/2

1.12 ALPROGRAMOK SZERVEZÉSE

A GOSUB és a RETURN kulcsszavak

Az 1.11 alfejezetben ismertetett függvény csak egyértékű lehet, és az utasítás hossza is korlátozó tényező, hiszen legfeljebb 80 karakter lehet. Gyakran csak több utasítás-sorban leírható algoritmust kell egy programon belül ismételtelen felhasználni és pl. az algoritmus részeredményei is szükségesek lehetnek a program további működéséhez.

A BASIC lehetőséget ad ilyen részprogramok, ún. *alprogramok* (szubrutinok) írására, amelyek abban a programban, amelyben létrehozzuk, tetszőleges számúszor felhasználhatók.

Az alprogram szerkezete: s1 (utasítás)

```

.
.
.
s2 RETURN

```

Az alprogram tehát egy meghatározott (s1) sorszámú utasítással kezdődik és egy s2 sorszámú RETURN utasítással végződik.

A RETURN kulcsszót karakterenként be kell írni, nem helyettesíthető a **RETURN** billentyű lenyomásával.

Az alprogram *hívóutasításának* alakja:

s GOSUB s1

amely az s1 sorszámú utasításra adja a vezérlést, majd amikor az utasítások végrehajtása során eléri az első **RETURN** utasítást, a vezérlés visszakerül a hívó GOSUB utasítást követő első utasításra.

Pl. a 19. program az 1000–1020-as, az 1100–1120-as és az 1200–1220-as utasítássorokban három alprogramot tartalmaznak. Az elsőt az 50-es, a 72-es, a 90-es, a másodikat a 95-ös, a 140-es, a 170-es és a 220-as, a harmadikat a 40-es, a 60-as és a 80-as sorszámú utasítások hívják.

```

10 PRINT "Q": REM *** 19. PROGRAM
12 :
14 REM ** GEOMETRIAI SZÁMITÁSOK: KOR, HENGER, GOMB
15 :
20 PRINT:PRINT " KOR:TERULET,KERULET"
22 PRINT " HENGER: TERFOGAT, FELSZIN"
30 PRINT " GOMB:TERFOGAT,FELSZIN"
32 PRINT " KISZAMITASA ALPROGRAMOKKAL.":PRINT
34 :
40 PRINT " KOR?(I/N)":GOSUB 1200
50 IF Q$="I" THEN INPUT " R=":R:GOSUB 1000:GOTO 150
60 PRINT " HENGER?(I/N)":GOSUB 1200
70 IF Q$>"I" THEN 80
72 INPUT " R=":R:INPUT " M=":M:GOSUB 1000:GOTO 180
80 PRINT " GOMB?(I/N)":GOSUB 1200
90 IF Q$="I" THEN INPUT " R=":R:GOSUB 1000:GOTO 100
95 GOSUB 1100
98 GOTO 10
100 VG=4/3*KT
110 FG=4*KT
120 PRINT:PRINT " GOMB TERFOGATA:":VG
130 PRINT " FELSZINE:":FG
140 GOSUB 1100:GOTO 10
150 PRINT:PRINT " KOR TERULETE:":KT
160 PRINT " KERULETE:":KK
170 GOSUB 1100:GOTO 10
180 VH=M*KT
190 FH=2*KT+M*KK
200 PRINT:PRINT " HENGER TERFOGATA:":VH
210 PRINT " FELSZINE:":FH
220 GOSUB 1100:GOTO 10
222 :
1000 KT=R*R*PI: REM *** 1. ALPROGRAM
1010 KK=2*R*PI
1020 RETURN
1022 :
1080 REM ** 2. ALPROGRAM:
1100 PRINT:PRINT " KEZDJUK ELOLOROL?(I/N)"
1102 PRINT " A VALASZ CSAK I VAGY N LEHET."
1104 INPUT Q$
1107 IF Q$>"I" AND Q$<"N" THEN 1100
1108 PRINT
1110 IF Q$="N" THEN PRINT " BEFEJEZVE.":PRINT "Q":END
1120 RETURN
1122 :
1130 REM ** 3. ALPROGRAM:
1200 PRINT " A VALASZ CSAK I VAGY N LEHET.":INPUT Q$
1210 IF Q$>"I" AND Q$<"N" THEN 1200
1220 RETURN

```

Ha a GOSUB utasításban olyan sorszámot adtunk meg, amely a programban nem fordul elő, akkor az értelmező

? UNDEF'D STATEMENT ERROR

üzenettel megszakítja a program futását.

Több alprogram ágyazható egymásba, azaz bármely alprogram hívhatja — önmagán kívül — az összes többit. Az alprogramok számát az korlátozza, hogy a tárban a visszatérési cím megőrzésére csak 256 byte-nyi terület áll rendelkezésre.

Ha a visszatérési címeknek már nem talál helyet az értelmező, akkor

? OUT OF MEMORY ERROR

üzenettel megszakítja a program futását.

Ha a visszatérési címet nem találja az értelmező, pl. a sorszámozás szerinti sorrendben és nem a GOSUB utasítás hatására hajtatta végre a program az alprogramot, akkor

? RETURN WITHOUT GOSUB ERROR IN *s*

üzenettel áll le a program, ahol *s* az elsőnek elért RETURN utasítás sorszáma.

FIGYELEM! Mivel a FOR és a GOSUB utasítások közös tárterületet használnak, ezért

```
2000 GOSUB 2222 : NEXT I
2222 FOR I=1 TO 500
2230 RETURN
```

szervezetű programrész esetén

? NEXT WITHOUT FOR ERROR IN 2000

hibaüzenetet kapunk, ui. a RETURN végrehajtása törli a FOR-ra vonatkozó információkat.

1.13 VEZÉRLÉSÁTADÓ UTASÍTÁSOK II.

Az ON...GOTO... és az ON...GOSUB kulcsszavak

A BASIC a többirányú elágazást az

ON *ak* GOTO *lista*
ill. ON *ak* GOSUB *lista*

alakú utasításokkal valósítja meg, ahol *ak* aritmetikai (logikai) kifejezés, a *lista* elemei pedig vesszővel elválasztott utasítássorszámok.

Az *ak* aritmetikai kifejezés egész részének ki kell elégítenie a

$$0 \leq \text{INT}(ak) \leq 255$$

feltételt. Az adott intervallumon kívül eső *ak* érték esetén az értelmező

? ILLEGAL QUANTITY ERROR IN *s*

üzenettel leállítja a program futását.

A lista annyiadik elemének megfelelő utasítássorra adódik át a vezérlés, amennyi az *ak* értéke. Pl.:

ON K GOTO 10, 100, 150

Ha K=1, akkor a GOTO 10, ha K=2, akkor a GOTO 100, ha K=3, akkor a GOTO 150 utasítás hajtódik végre. Hasonlóan az

ON K GOSUB 1000, 1200, 1400

utasítás K=1 esetén a GOSUB 1000; K=2 esetén a GOSUB 1200, K=3 esetén a GOSUB 1400 utasítással azonos hatású.

Ha K=0 vagy K nagyobb a lista elemeinek számánál, akkor a program hibajelzés nélkül a következő sorral folytatódik.

Az ON kezdetű vezérlésátadó utasításban a GOTO nem írható szóközzel, azaz GO TO alakban, mert ilyen esetben

? SYNTAX ERROR IN *s*

üzenettel leáll a program.

Ha a lista elemei között a programban nem létező utasítássorszám van és a vezérlés erre a sorszámmra adódik át, akkor a

? UNDEF'D STATEMENT ERROR IN *s*

üzenetet kapjuk és a program futása megszakad (*s* az ON utasítás sorszáma).

Pl.:

```
2 PRINT "U": REM*** 20.PROGRAM
3 :
4 PRINT "ON...GOTO/GOSUB...VEZERLESATADAS"
5 PRINT:PRINT " ON K GOTO 30,12,20":PRINT
6 :
8 FOR K=1 TO3
10 ON K GOTO 30,12,20
12 PRINT"GOTO 12 , HA K=2":GOTO35
20 PRINT"GOTO 20 , HA K=3":GOTO35
30 PRINT"GOTO 30 , HA K=1":GOTO35
35 NEXT K
38 J=1
39 PRINT:PRINT " ON J GOSUB 60,90,80":PRINT
40 ON J GOSUB 60,90,80:J=J+1:IF J<=3THEN40
50 GOTO150
60 PRINT" GOSUB 60 , HA J=1":RETURN
80 PRINT" GOSUB 80 , HA J=3":RETURN
90 PRINT" GOSUB 90 , HA J=2":RETURN
100 END
150 PRINT:PRINT " A 170-ES SORSZAMU UTASITAS HAROM"
152 PRINT" IF...THEN...SZERKEZETNEK MEGFELELOEN "
154 PRINT" VALASZTJA KI A KOVETKEZO UTASITAS"
156 PRINT" SORSZAMOT.":PRINT
160 FOR M=1TO5STEP2
170 ON -(M=5)-2*(M=3)-3*(M<2) GOTO 200,220,240
200 PRINT"GOTO 200 , MIVEL M=5-RE AZ AK=1":GOTO 250
220 PRINT"GOTO 220 , MIVEL M=3-RA AZ AK=2":GOTO250
240 PRINT"GOTO 240 , MIVEL M=1-RE AZ AK=3":GOTO250
250 NEXT M
260 END
```


Az elágazást a következő kifejezés értékei vezérlik:

$$AK = -(M=5) - 2*(M=3) - 3*(M<2) = \begin{cases} 1, & \text{ha } M=5 \\ 2, & \text{ha } M=3 \\ 3, & \text{ha } M<2 \end{cases}$$

1.14 BEVITELI UTASÍTÁSOK II.

A DATA, a READ, a RESTORE és a GET kulcsszavak

Számos műszaki-gazdasági feladat megoldási eljárásához a programban konstansokat kell tárolni. Ezek a konstansok lehetnek az eljárás részei, akkor a program használata során nem változnak és lehetnek ritkán változó vagy bővülő adatok.

A BASIC a konstansok programbeli tárolására a

s DATA lista

alakú utasítást használja. A *lista* elemei vesszővel elválasztott konstansok.

A listában számkonstansok és füzérkonstansok vegyesen szerepelhetnek.

A füzérkonstansokat (szövegeket) csak akkor kell idézőjelek közé tenni, ha a szöveg vesszőt is tartalmaz.

A DATA lista konstansait az

s READ változólista

alakú utasítás adja át a változólista változóinak. A változólista elemei egymástól vesszővel elválasztott, tetszőleges típusú egyszerű vagy tömbváltozók lehetnek. A READ utasítás változólistájában legalább egy változónak lennie kell, mert különben

? SYNTAX ERROR

üzenettel leáll a program.

A READ a változólista elemeinek a DATA listaelemeket balról jobbra haladva folyamatosan adja át. Ha a READ utasítás több adatot akar beolvasni, akkor

? OUT OF DATA ERROR IN *s*

üzenettel megáll a program.

Ha a READ lista utolsó eleme után is vesszőt teszünk, akkor a

? SYNTAX ERROR IN *s*

üzenettel áll meg a program.

Ha a DATA lista utolsó eleme után teszünk vesszőt, akkor nincs hibajelzés; az értelmező a következő elemet 0 értékűnek veszi, ha a READ lista következő változója valós típusú, ha pedig füzér típusú, akkor üres füzér ("") tartalmú lesz.

Ha a READ egész vagy valós változójú listaeleme füzérkonstansot kap a DATA listából, akkor a

? SYNTAX ERROR IN *s*

üzenettel áll le a program, ahol *s* a DATA utasítás sorszáma.

Egy programban több READ és DATA utasítás lehet. A READ utasítások a DATA listákat az egymás után következő DATA utasításokból folyamatosan balról jobbra olvassák be, s így, ha valamelyik DATA listáját egy READ utasítás nem mérítette ki, akkor a következő READ utasítás onnan folytatja az adatok beolvasását, ahol az előző READ befejezte.

Ha a program legelső DATA listájának első elemétől kell ismét valamely READ utasítással adatot beolvasni, akkor az

s RESTORE

utasítást írjuk be a programba, a szóban forgó READ előtt.

Pl.:

```
5 PRINT "": REM*** 21.PROGRAM
6 :
7 PRINT:PRINT " A READ, DATA, RESTORE UTASITASOK "
8 PRINT " EREDMENYEINEK SZEMLELTETESE:" :PRINT
9 :
10 READ R,T,Y,C#:REM** 100-AS DATA 4. ELEME
30 PRINT R,T,Y,C#:PRINT
50 READ R#,T#,C#:REM** 120-AS DATA 3.ELEME:PRINT
60 PRINT "R#=";R#,"T#=";T#,"C#=";C#:PRINT
70 RESTORE:REM** 100-AS DATA-RA
80 READ A,S:REM** 100-AS DATA 2.ELEME
90 PRINT A,S:PRINT
92 READ G:REM** 100-AS DATA 3. ELEME
94 PRINT G:PRINT
95 READ G#,F#,P#:REM** 100-AS DATA 4. ES
96 REM** 120-AS DATA 1., 2. ELEME
98 PRINT G#,F#,P#
100 DATA 2,3,4,0
120 DATA BIRKA,KECSKE,SZARVASMARHA
130 END
```

A billentyűzet karaktereinek beolvasására az

s GET X, ill *s GET XS*

utasítás való.

A GET utasítással egyetlen byte-ot tudunk beolvasni a billentyűzetről. Az értelmező az egymás után leütött billentyűk karaktereit az ún. billentyűzetpufferben tárolja, de csak legfeljebb 10-et. Ezt a 22. programmal kipróbálhatjuk. A program 30-as sorszámú utasítása addig várakozik, amíg le nem ütünk egy billentyűt. Az 50—56-os sorszámú utasítások lehetővé teszik, hogy kb. 16—18 billentyűt gyorsan leüssünk. Azt láthatjuk, hogy mindig az első tíz leütött billentyűnek megfelelő karakter íródik ki, a többi nem tárolódott, mert azoknak a pufferben már nem volt hely.


```

10 PRINT "Q": REM*** 22. PROGRAM
12 PRINT
13 :
20 PRINT TAB(6) " GET UTASITAS HATASA.":PRINT
22 PRINT " EGY BILLENTYU LEUTESE UTAN GYORSAN"
24 PRINT " USSON LE 10-NEL TOBBET A PROGRAM"
25 PRINT " LEALLASAIG!":PRINT
26 :
30 GET Q$: IF Q$="" THEN 30:REM** VARAKOZAS
40 PRINT Q$
50 FOR I=1 TO 4000:NEXT I
52 FOR J=0 TO 18
54 GET Z$:PRINT SPC(2)Z$;
56 NEXT J
57 PRINT:PRINT " CSAK 10 KARAKTER JELENIK MEG!"
60 STOP

```

Megjegyzés: az INPUT utasításhoz hasonlóan a GET utasítás sem használható számológép-üzem módban. Ha sorszám nélkül írjuk le ezeket az utasításokat, akkor

? ILLEGAL DIRECT ERROR

üzenetet kapunk.

Ha a GET numerikus típusú változójának nem numerikus típusú karaktert küldünk valamely billentyű leütésével, akkor

? SYNTAX ERROR

üzenetet küld az értelmező. Pl.:

```

10 GET R: IF R=0 THEN 10
20 PRINT R

```

A program a várakozása alatti nem számkonstans tartalmú billentyű leütése esetén

? SYNTAX ERROR

üzenetet ad. (Kivételek: CTRL, C=, SHIFT, RESTORE, +, - és az E billentyű.)

Célszerű füzerváltozót használni a GET után, és számkonstans bevitele esetén, — ha arra a programban számjegyként lesz szükség — a VAL függvényértékét képezzük (l. az 1.15 alfejezetben).

```

10 PRINT "Q": REM*** 23. PROGRAM
12 :
20 PRINT:PRINT " A GET UTASITASSAL KAPOTT"
22 PRINT " EREDMENYEK SZEMLELTETESE: USSON LE"
24 PRINT " NEHANY SZAMJEGY ES BETU BILLENTYUT."
25 PRINT
26 PRINT " A PROGRAMBOL A RUN/STOP BILLENTYU"
28 PRINT " LEUTESEVEL LEPHET KI."
29 :
30 GET A$: IF A$="" THEN 30
70 PRINT "A$=";A$
80 A=VAL(A$):REM** NUMERIKUS ERTEKKE ALAKITAS
90 PRINT "A=";A:PRINT " A*3=";A*3
100 GOTO 60

```

A program több betű- és számjegyekarakter bevitelével szemlélteti az elmondottakat. A programot **RUN/STOP** billentyű leütésével állíthatjuk le. (A betűkarakterek VAL függvényértéke=0.)

1.15 A FÜZÉR (STRING) FÜGGVÉNYEK

A LEN, az STR\$, a VAL, a CHR\$, az ASC, a LEFT\$, a RIGHT és a MID\$ kulcsszavak

A füzérkonstansokkal és a füzerváltozókkal végzendő műveleteket, valamint az adatfeldolgozás eljárásainak kényelmesebb végrehajtását az ún. *füzérfüggvények* teszik lehetővé. A BASIC nyelv C 64-es számítógépen alkalmazható változata nyolc füzérfüggvényt definiál. A füzérfüggvények számológép- és tároított programú üzemmódban egyaránt használhatók.

A LEN(A\$), ill. LEN (füzér) függvény

Egész típusú aritmetikai függvény az A\$="karakterek", ill. a füzér összes karaktereinek számát képezi, beleszámítva a szóköz és a nem kinyomtatható karaktereket is. Pl.

```

10 PRINT "Q": REM*** 24. PROGRAM
20 PRINT:PRINT " LEN FUGGVENYERTEK SZEMLELTETESE:":PRINT
30 A$="EGYETEM: ":PRINT A$;
40 B$=" EGYETEM: ":PRINT B$;
50 PRINT LEN(A$),LEN(B$)
60 PRINT LEN "EGYETEM: ",LEN " EGYETEM: "
70 END

```

A képernyőkép:

LEN FUGGVENYERTEK SZEMLELTETESE:

EGYETEM:	EGYETEM:
10	12
10	12

READY.

Ha az argumentum nem füzér típusú, akkor

? TYPE MISMATCH ERROR IN s

hibaüzenettel áll le a program.

Az STR\$(ak) függvény

Füzér típusú függvény, az *ak* aritmetikai értéket füzér típusúvá alakítja. Figyelembe veszi a számok utáni automatikus szóközt és pozitív szám esetén a + jel helyett értelmezett szóközt is. Az STR és a \$ között nem lehet szóköz, mert ilyen esetben az értelmező az ST\$(ak) fiktív tömbelemet vizsgálja, s ha ilyet nem talál, akkor

? BAD SUBSCRIPT ERROR IN *s*

üzenettel áll le a program, ha *ak* > 10.

Ha az argumentum nem aritmetikai kifejezés (változó, konstans) akkor, a

? TYPE MISMATCH ERROR IN *s*

hibaüzenettel leáll a program.

Pl.:

```
10 PRINT"Ü": REM*** 25.PROGRAM
20 PRINT:PRINT" AZ STR$ FUGGVENY ERTEKKEPZESENЕК "
22 PRINT" SZEMLELTETESE:" :PRINT
30 V=1986:R=1.986E3
40 V$=STR$(V):R$=STR$(R)
50 PRINT " V=";V;" V$=";V$
60 PRINT " R=1.986E3";" R$=";R$
70 END
```

A képernyőkép:

AZ STR\$ FUGGVENY ERTEKKEPZESENЕК
SZEMLELTETESE:

V= 1986 V\$= 1986
R=1.986E3 R\$= 1986

READY.

A VAL(A\$), ill. a VAL(füzér) függvény

A VAL aritmetikai függvény, az argumentumként adott füzérkifejezés numerikus értékét számítja ki. A VAL függvény balról jobbra haladva addig végzi a füzér karaktereinek átalakítását, amíg a

+ - E (csak számkonstansban). 0 1 2 ... 9

karakterektől eltérő karaktert nem talál, azaz az első számrészként nem értelmezhető karakterig.

Ha a füzér első karaktere a felsoroltaktól eltér, akkor 0 értéket vesz fel a VAL függvény. (Az E első karakter, vagy nem szabványosan írt számok, pl. +5 esetében is.)

Ha az argumentum nem füzérkifejezés, akkor

? TYPE MISMATCH ERROR IN *s*

hibaüzenetet kapjuk, s a program leáll.
(Alkalmazását l. a 23. programban.)

A CHR\$(ak) függvény

Füzér típusú függvény, az *ak* aritmetikai kifejezés egész értékével megegyező, ASCII kóddal (l. a Függelékben) meghatározott karaktert adja eredményül, ha

$0 \leq \text{INT}(ak) \leq 255$

Ha az $\text{INT}(ak) < 0$ vagy $\text{INT}(ak) > 255$, akkor az értelmező

? ILLEGAL QUANTITY ERROR IN *s*

üzenettel állítja meg a programot.

Ha a CHR és a \$ közé szóköz kerül, akkor a CH\$(ak) fiktív tömb vizsgálata miatt *ak* > 10 esetén

? BAD SUBSCRIPT ERROR IN *s*

üzenettel áll le a program.

A CHR\$ lehetővé teszi, hogy a programba speciális szerkesztőkaraktereket helyezzünk el.

Pl.:

```
10 PRINT"Ü": REM*** 26.PROGRAM
12 :
20 PRINT:PRINT" * CHR$ FUGGVENY EREDMENYEINEK "
22 PRINT" SZEMLELTETESE."
23 :
30 PRINT" PROGRAMBOL VALTAS"
32 PRINT" NAGYBETUK/KISBETUK ALLASRA:"
40 PRINT:PRINT" PRINTCHR$(14)CHR$(8) HATASA:"
50 PRINTCHR$(14):REM** NAGYBETUK/KISBETUK VALTAS
60 PRINTCHR$(8): REM** A C= ES A SHIFT BILLENTYUK
62 REM** EGYUTTES HATASA KIKAPCSOLVA
70 PRINT" * SHIFT BILLENTYUT HASZNALVA NAGYBETU-"
80 PRINT" KET IS IRHATUNK." :PRINT
90 PRINT" * I - - - I I \ ' A B C D E F G H I J"
100 PRINT:PRINT" * / * I I \ , ES A KISBETU."
110 FOR I=1 TO 6000:NEXT I
120 PRINT:PRINT" A PRINTCHR$(142) NAGYBETUK/GRAFIKUS JELEK"
122 PRINT" ALLASRA VALT, A"
130 PRINT:PRINT" CHR$(9) VISSZAKAPCSOLJA A "
132 PRINT CHR$(34)"SHIFT"CHR$(34)" ES " ;
140 PRINTSPC(3)CHR$(34)"C="CHR$(34)" BILLENTYUKET."
150 FOR I=1 TO 6000:NEXT I
160 PRINT:PRINT" ATVALTAS "CHR$(34)" C="CHR$(34);
162 PRINT" ES "CHR$(34)"SHIFT"CHR$(34);
170 PRINT" BILLENTYUK LENYOMASAVAL."
180 PRINTCHR$(142)CHR$(9)
190 PRINT"CHR$(34) IDEZOJEL ELHELYEZESE SZOVEGBEN."
```


A CHR\$(0) jelentése üres karakter, a nyomtatványban nem jelenik meg.
Pl.:

```
10 PRINT "A": REM*** 27.PROGRAM
12 :
15 PRINT:PRINT " A CHR$(0), A CHR$(17), A CHR$(29), "
17 PRINT " A CHR$(5) ES CHR$(154) HATASAI. "
18 :
20 A$="19"+CHR$(0)+"86"
30 V=VAL(A$)
40 PRINT:PRINT:PRINT "VAL(A$)=";V
50 Z=LEN(A$):REM*** FUZER HOSSZA
60 PRINT "LEN(A$)=";Z
70 PRINT "A$=";A$
90 FOR I=1 TO 2000:NEXT I
90 PRINTCHR$(17)CHR$(29)CHR$(29)"ALMA"
100 PRINT:PRINTCHR$(5)" EGY LEPESE LEFELE, "
102 PRINT " KET LEPESE JOBBRA. "
110 FOR I=1 TO 3000:NEXT
120 PRINT:PRINT:PRINTCHR$(154)" BEFEJEZVE. "
130 END
```

A programban a CHR\$(17) a kurzort egy sorral lejjebb, a CHR\$(29) egy oszlop-jal jobbra viszi, a CHR\$(5) a karakterek színét fehérre, a CHR\$(154) pedig ismét világoskékre változtatja.

Az ASC(A\$), ill az ASC(„füzér”) függvény

Az ASC aritmetikai függvény, amely az A\$=„karakterek”, ill. a füzér kifejezés első karakterének ASCII kódját (l. a Függelékben) adja, amely 0—255 közé eső szám.

Ha az argumentum nem füzér vagy füzérváltozó, ill. a hossza 0, akkor

? ILLEGAL QUANTITY ERROR IN s

hibaüzenettel áll le a program.

Ha nem akarjuk, hogy A\$=„” esetén hibajelzéssel leálljon a program, akkor a CHR\$(0) hozzákapcsolásával, (amely 1 hosszúságú üres karaktert jelent, de mint már említettük a nyomtatásban nem jelenik meg) a ASC(A\$+CHR\$(0)) argumentumot használunk az ASC(A\$) helyett.

Ha az argumentum nem füzér típusú, akkor

? TYPE MISMATCH ERROR IN s

üzenettel áll le a program.

Pl.:

```
10 PRINT "A": REM 29.PROGRAM
12 :
20 PRINT:PRINT " AZ ASC FUGGVENY EREDMENYE INEK "
22 PRINT " SZEMLELTETESE. "
28 :
30 PRINT:PRINTASC( "ALMA">SPC(2))="A"
32 PRINT ASC( "A">SPC(2))="A",ASC( "E1927">SPC(2))="E"
40 W$=" "
50 PRINT:PRINTASC( W$+CHR$(0))
60 END
```

A LEFT\$(A\$,ak), ill. LEFTS (füzér, ak) függvény

(0≤INT(ak)≤255)

A LEFTS kétváltozós füzérfüggvény, ahol az ak aritmetikai kifejezés helyett szám-konstans is állhat. A függvény az A\$=„karakterek”, ill. a füzér karaktereinek első INT(ak)-val adott számú karakteréből egy új füzért képez.

Az ak aritmetikai kifejezés negatív értéke esetén

? ILLEGAL QUANTITY ERROR IN s

hibaüzenettel áll le a program.

Ha ak egész értéke nagyobb, mint az A\$, ill. a füzér karaktereinek száma, akkor hibajelzés nélkül az összes karaktert adja vissza eredményül.

Ha ak=0, akkor a függvény értéke a „”, vagyis az üres füzér.

Pl.:

```
10 PRINT "A": REM*** 29.PROGRAM
20 :
30 PRINT:PRINT " A LEFT$ FUGGVENY EREDMENYEINEK "
32 PRINTTAB(8)" SZEMLELTETESE. "
40 PRINT "(D$= JO NAPOT!)":PRINT
42 :
50 PRINT:PRINT " USSON LE PL. 1-12 KOZOTT HAROM"
52 PRINT " SZAMOT VESSZOVEL ELVALASZTVA! "
60 PRINT:INPUT N,M,V:PRINT
70 D$=" JO NAPOT! "
80 PRINT LEFT$( " JO NAPOT! ".N)SPC(3)"N=";N
90 PRINT LEFT$(D$,M)SPC(3)"M=";M
100 PRINT LEFT$(D$,V)SPC(3)"V=";V:PRINT
110 PRINT " USSON LE 1-2 BETU-, MAJD SZAMKARAKTERT! "
112 PRINT
120 FOR L=1 TO 1000:NEXT
130 GET I$:I=VAL(I$):FOR K=1 TO 1500:NEXT
140 PRINT "I$=";I$SPC(3);
142 Q4$=" LEFT$(D$,I) ="
144 Q5$=LEFT$(D$,I)
150 IF Q5$=" " THEN PRINT "I=";I:PRINT Q4$;Q5$:GOTO 130
160 PRINT "I=";I;Q4$;Q5$
170 PRINT:PRINTTAB(16)"VEGE":END
```

A RIGHT\$(A\$,ak), ill. a RIGHTS(„füzér”, ak) függvény

(0≤INT(ak)≤255)

A RIGHTS kétváltozós füzérfüggvény, ahol az ak aritmetikai kifejezés helyett szám-konstans is állhat. A függvény az A\$=„karakterek”, ill. a füzér karaktereinek utolsó karakterétől számított INT(ak)-val adott számú karakteréből egy új füzért képez.

Az ak negatív értéke esetén

? ILLEGAL QUANTITY ERROR IN s

hibaüzenettel áll le a program.

Ha ak egész értéke nagyobb, mint $A\$$, ill. a füzér karakterszáma, akkor kijelzés nélkül az összes karaktert adja vissza eredményül.

Ha $ak = 0$, akkor a függvény a "", vagyis az üres füzér értéket veszi fel.

Pl.:

```
10 PRINT "Q": REM *** 30. PROGRAM
12 :
20 PRINT: PRINT "A RIGHT$ FUGGVENY EREDMENYEINEK"
22 PRINT "      SZEMLELTETESE."
24 :
30 PRINT: PRINT " (D$= JO NAPOT!)" : PRINT: PRINT
40 PRINT " USSON LE PL. 1-12 KOZOTT HAROM SZAMOT "
42 PRINT " VESSZOVEL ELVALASZTVA!" : PRINT
44 :
45 :
50 INPUT N,M,V: PRINT
60 D$=" JO NAPOT!"
70 PRINT RIGHT$( " JO NAPOT!" ,N) SPC(3) "N=";N
80 PRINT RIGHT$(D$,M) SPC(3) "M=";M
90 PRINT RIGHT$(D$,V) SPC(3) "V=";V
100 PRINT: PRINT " USSON LE 1-2 BETU-, MAJD EGY "
102 PRINT " SZAMKARAKTERT!" : PRINT
110 FOR L=1 TO 1000: NEXT
120 GET I$: I=VAL(I$): FOR K=1 TO 1500: NEXT
130 PRINT " I$="; I$ SPC(3);
132 Q4$="RIGHT$(D$,I) ="
134 Q5$="RIGHT$(D$,I) ="
140 IF Q5$="" THEN PRINT " I="; I; : PRINT Q4$; Q5$: GOTO 120
150 PRINT: PRINT " I="; I; Q4$; Q5$
152 PRINT TAB(14); " (D$= JO NAPOT!)"
160 PRINT: PRINT TAB(16); "VEGE": END
```

A MID(A$, ak1, ak2)$, ill. a MID(füzér, ak1, ak2)$ függvény

$0 < INT(ak1) \leq 255$, $0 \leq INT(ak2) \leq 255$.

A $MID\$$ két vagy háromváltozós füzérfüggvény, az $A\$$ ="karakter", ill. a füzér karaktereiből állít elő új füzért, mégpedig az $INT(ak1)$ -edikről az $INT(ak2)$ -edikkel megadott számú karakter felhasználásával. Ha az $INT(ak1)$ nagyobb, mint az $A\$$, ill. a füzér karakterszáma, vagy az $INT(ak2)$ értéke 0, akkor a $MID\$$ függvény a "", vagyis az üres füzér értéket veszi fel. Ha az $INT(ak2)$ nagyobb, mint az első argumentum karaktereinek $INT(ak1)$ -től számlált karakterszáma, akkor a függvény értéke az $INT(ak1)$ -től az összes karakter lesz.

Ha a harmadik argumentum, $ak2$ hiányzik, akkor a függvény $ak2=255$ -nek megfelelően működik.

Ha az első argumentum karakterszáma 255-nél nagyobb, ill. ha a második vagy a harmadik argumentum a megadott intervallumon kívüli értéket vesz fel, akkor a

? ILLEGAL QUANTITY ERROR IN s

üzenettel áll le a program.

Pl.:

```
10 PRINT "Q": REM *** 31. PROGRAM
12 :
20 PRINT: PRINT " A MID$ FUGGVENY EREDMENYEINEK "
22 PRINT "      SZEMLELTETESE."
30 PRINT: PRINT " (D$= JO NAPOT!)" : PRINT: R=1
32 :
40 PRINT: PRINT " USSON LE PL. 0-12 KOZOTT KET"
42 PRINT " SZAMOT VESSZOVEL ELVALASZTVA!" : PRINT
44 :
45 :
50 INPUT N,M: ON R GOTO 60,80
60 D$=" JO NAPOT!"
70 PRINT MID$( " JO NAPOT!" ,N,M) SPC(3) "N=";N; "M=";M
72 R=R+1: GOTO 50
80 PRINT MID$(D$,N,M) SPC(3) "N=";N SPC(2) "M=";M
90 PRINT: PRINT " USSON LE 1-2 BETU-, MAJD "
92 PRINT "SZAMKARAKTERT!" : PRINT
100 FOR L=1 TO 1000: NEXT
110 GET I$: I=VAL(I$): FOR K=1 TO 1500: NEXT
120 PRINT " I$="; I$ SPC(3);
122 Q4$="MID$(D$,3,1) ="
124 Q5$="MID$(D$,3,1) ="
130 IF Q5$="" THEN PRINT " I="; I; : PRINT Q4$; Q5$: GOTO 110
140 PRINT: PRINT " I="; I; "MID$(D$,1,I) =" ; MID$(D$,1,I)
150 PRINT: PRINT TAB(16); "VEGE": END
```

Elemezzük a következő programot a $MID\$$, a VAL , az ASC függvények alkalmazása szempontjából!

```
10 PRINT "Q": REM *** 32. PROGRAM
12 :
20 PRINT: PRINT " KONVERTALAS 8 BITES BINARIS SZAMROL "
22 PRINT "      DECIMALIS SZAMRA."
30 PRINT: INPUT " KEREM A BINARIS SZAMOT:" ; B$
32 PRINT: L$=LEN(B$)
34 S4$=" EZ NEM 8 HELYI ERTEKES SZAM!"
40 IF L$ < 8 THEN PRINT S4$: GOTO 120
50 D=0: L=0
60 FOR K=8 TO 1 STEP -1: L=L+1
70 V$=MID$(B$,L,1)
72 S5$=" EZ NEM BINARIS SZAM!"
80 IF ASC(V$) < 48 OR ASC(V$) > 49 THEN PRINT S5$: GOTO 120
90 D=D+VAL(V$)*2^(K-1)
100 NEXT K
110 PRINT: PRINT TAB(10); "DECIMALIS SZAM="; D
120 B$="": PRINT: PRINT "BEFEJEZTE?(I/N)"
130 GET Q$: IF Q$="" THEN 130
140 IF Q$="N" THEN 30
150 PRINT: PRINT " BEFEJEZVE."
160 PRINT "Q"
170 END
```


1.16 A MÁGNESLEMEZ, A MÁGNESSZALAG ÉS A NYOMTATÓEGYSÉG HASZNÁLATA

Az OPEN, a PRINT, a SAVE, a VERIFY, a LOAD, a VALIDATE, a CLOSE és a CMD kulcsszavak

Ezekkel a kulcsszavakkal kapcsolatban csak a legszükségesebbeket említjük meg, bővebb ismertetésükre a 4. és az 5. fejezetben még visszatérünk.

A programot megírása, ellenőrzése, próbafuttatása, szükséges javítása után — rendszerint — a főtárból a C 64-es konfigurációjához tartozó lemezegység vagy kazettás egység közreműködésével mágneslemezre vagy mágnesszalagra másoljuk át, hogy a későbbiekben azt ismételtelen felhasználhassuk, ill. a nyomtatóegységgel olvasható formában papírra nyomtatjuk. Mivel a számítógép kikapcsolása törli a tárban levő programot, így kikapcsolás előtt minden megőrzendő programot valamelyik külső tárolóra ki kell menteni.

A lemezegységhez rendszeresített lemezt, ha az új, csak akkor tudjuk használni, ha a lemezt a lemezegységbe helyezve pl. az

```
OPEN15,8,15,"N:lemeznév,AZ":CLOSE15
```

paranccsal előkészítjük, formázzuk (a N a NEW szó rövidítése). Kb. másfél perc múlva, amikor leáll az egység lemezmeghajtója, elalszik a piros jelzőfény, a lemez használható.

A lemeznév legfeljebb 16 karakternyi lehet. Ha a megjelölt lemezegység (8) nincs bekapcsolva akkor

```
? DEVICE NOT PRESENT
```

üzenetet, ha a parancscsatorna (15) már megnyitott volt, akkor

```
? FILE OPEN
```

üzenetet kapunk.

A formázás során a lemezen a lemez-tartalomjegyzék fejléce elkészült, amely egy 0 karakter után idézőjelek között a lemeznév és a két karakteres AZ azonosító szöveget tartalmazza, valamint a 2A karaktereket, amelyek a VC—1541-es típusú lemezegység kódjai. A tartalomjegyzék mindig feltünteteti egy külön sorban, hogy mennyi a szabad blokkok száma.

Az üres lemezen 664 blokk tárolására van szabad hely. Egy blokk 256 byte-ból áll.

```
A
```

```
LOAD "$",8
```

parancs és a LIST parancs végrehajtása után a képernyőn megjelenik a lemez tartalomjegyzéke.

Pl. az OPEN 15,8,15,"N:OKTATAS1,01":CLOSE15 parancs, a LOAD"\$",8 és a LIST parancsok végrehajtása után a következőket láthatjuk a képernyőn:

```
OPEN15,8,15,"N:OKTATAS1,01":CLOSE15
```

```
READY.  
LOAD"$",8
```

```
SEARCHING FOR $  
LOADING  
READY.  
LIST
```

```
0 "OKTATAS1" 01 2A  
664 BLOCKS FREE.  
READY.
```

A parancsok és a READY. közötti két sorban a lemez tartalomjegyzéke olvasható a képernyőn.

A tárban levő programot a lemezre a

```
SAVE"programnév",8
```

parancs végrehajtásával írathatjuk ki. A programnév legfeljebb 16 karakternyi lehet.

Ha ellenőrizni akarjuk, hogy a tárban levő programmal azonos tárolódott-e a lemezen is, akkor a

```
VERIFY"programnév",8
```

parancsot adjuk ki, amelynek

```
OK
```

válasza igazolja az azonosságot és

```
? VERIFY ERROR
```

üzenete az eltérést.

A lemeztől a tárba a programot a

```
LOAD"programnév",8  
ill. LOAD"programnév",8,1
```

beviteli paranccsal írjuk be. Az első parancs a programot a tár BASIC területének elejére tölti, a második parancs a programot a tár azon területére tölti be, ahonnan a lemezre másolta.

Az előzőleg lemezre másolt program, vagy a rendszer bekapcsolása után egy lemez első programja a

```
LOAD"*",8
```

paranccsal is betölthető a tárba.

A BIRKANAPLO nevű program a

```
LOAD"B*",8
```

paranccsal is hibátlanul betölthető, ha az az első B-vel kezdődő programnév a lemez tartalomjegyzékében. Ez természetesen más kezdőbetűvel beírt program bevitelére is igaz.

A programnév egyes betűi helyett kérdőjel (?) is írható, pl. a

```
LOAD"B???A",8
```

parancs betölti a tárbba a lemezen található első olyan nevű programot, amelynek öt karaktere közül az első betű B, az utolsó pedig A.

Ha a LOAD parancsban olyan programnévre hivatkozunk, amilyen a lemezen nem található, akkor

? FILE NOT FOUND

üzenetet, ha pedig a megjelölt lemezegység (8) nincs bekapcsolva, akkor

? DEVICE NOT PRESENT

üzenetet kapunk.

A lemezzről a tárbba betöltött program módosítása utáni, az eredeti programnévvel való lemezzre másolása a

```
SAVE"@0:programnév",8
```

paranccsal végezhető el (@ a kapcsolójel). (Ezt a parancsot a DOS ROM szoftverének hibája miatt nem célszerű használni.)

Ha olyan programnéven akartunk másolni a tárból lemezzre, amilyen nevű program már van a lemezen, akkor

? FILE EXISTS

hibajelzést kapunk.

A lemezzről olyan programot, amelyre már nincs szükség, az

```
OPEN15,8,15,"S:programnév": CLOSE 15
```

parancs végrehajtásával *törölhetünk* (az S a SCRATCH szó rövidítése).

A lemezkezelés közben elkövetett hibák után, a bekapcsolás utáni alapállapotot az

```
OPEN15,8,15, "I": CLOSE 15
```

parancs végrehajtásával érhetjük el. (I az INITIALIZE szó rövidítése).

Ha egy lemez programjai közül már többet töröltünk (pontosan: törléskor csak a program neve törölődik a tartalomjegyzékből), akkor a törölt nevű programok blokkjainak felszabadítását, a lemez *újraszervezését* az

```
OPEN15,8,15,"V":CLOSE 15
```

paranccsal végezhetjük el (a V a VALIDATE kulcsszó rövidítése).

A Commodore 64-es konfigurációjához tartozó C2N kazettás egységben levő mágnesszalagra a

```
SAVE"programnév"
```

paranccsal másolhatunk. Az értelmező a képernyőre kiírja a

```
PRESS RECORD PLAY ON TAPE
```

üzenetet, s a megfelelő kezelőgombok lenyomása utána a tárból a kazetta mágnesszalagjára másolódik a program, miközben a képernyőn

```
OK  
SAVING programnév
```

felirat látható.

A mágnesszalagra másolt program azonosságát a tárbeli programmal a

```
VERIFY"programnév"
```

parancs végrehajtásával ellenőrizhetjük, ha előtte a szalagot a program elejére állítjuk. Az

```
OK  
READY,
```

üzenet igazolja az azonosságot, a

```
VERIFY ERROR  
READY.
```

pedig a hibát jelzi.

A program bevitele mágnesszalagról a

```
LOAD "programnév"
```

paranccsal indul, majd a képernyőn a

```
PRESS PLAY ON TAPE
```

üzenet megjelenése után be kell nyomni a PLAY gombot.

A képernyőn

```
OK  
SEARCHING FOR programnév  
FOUND programnév  
LOADING
```

feliratok jelennek meg, majd a READY.

A nyomtatón az

```
OPEN 1,4:CMD 1:LIST,  
ill. OPEN 1,4,7:CMD 1:LIST
```

parancsok végrehajtásával a tárból kiírthatjuk a programot.

A második alakban az OPEN harmadik helyén levő 7-es (másodlagos cím) kis- és nagybetűs karakterkészletre való váltást ír elő, ha ez elmarad, ill. 0, akkor az első alaknak a hatására a nagybetűk/grafikus jelek írásmód valósul meg.

A nyomtatás befejezése után az adatcsatornát a

```
PRINT#1 :CLOSE 1
```

paranccsal zárjuk le.

A CMD 1 parancs a képernyőre vonatkozó LIST, PRINT parancsok vagy utasítások végrehajtó eszközeként az OPEN-ben kijelölt eszközt jelöli meg.

Az OPEN 1,4 megnyitja az 1-es adatcsatornát (kijelöli az 1-es file-számot) és a 4-es egység számú eszközt (ami általában a nyomtató) jelöli ki a parancs végrehajtására, amit most a LIST ír elő.

Programból szöveget, vagy a változók értékeit pl. a

```
s OPEN1,4
.....
s1 PRINT#1,"szöveg", V1,V2,...
.....
s2 PRINT# 1:CLOSE1
```

utasításokkal nyomtathatjuk ki.

A PRINT# utasítás listájában a nyomtató vezérlésére a következő vezérlőkódokat használhatjuk az MPS 801-es mátrixnyomtató esetén:

Áttérés kis- és nagybetűs karakterkészletre: CHR\$(17)

Áttérés nagybetűk/grafikus jelek állásra: CHR\$(145)

Áttérés grafikus üzemmódra: CHR\$(8)

Soremelés nyomtatás után: CHR\$(10)

Kocsi vissza, soremelés: CHR\$(13)

Áttérés duplaszélességű karakter módra: CHR\$(14)

Visszatérés duplaszélességű karakter módból: CHR\$(15)

A nyomtatás kezdőoszlopának meghatározása: CHR\$(16)"nn"

Negatív (inverz) karakterek nyomtatása: CHR\$(18)

Visszatérés inverz üzemmódról: CHR\$(146)

Grafikus jel ismétlése: CHR\$(26)

A Commodore 1526-os kétirányú mátrixnyomtatóhoz használható további vezérlő-jelek:

Lapdobás: CHR\$(12)

Lapüzemmód megszüntetése: CHR\$(19)

Lapüzemmód beállítása: CHR\$(147)

Kocsi vissza, soremelés nélkül: CHR\$(141)

Szóközlírás füzérek elejére: CHR\$(160)

Felhasználó által definiált karakterek kiírása: CHR\$(254)

Duplaszélességű (8×16-os) karakterek nyomtatása: CHR\$(1)

Normál (8×8-as) karakterek nyomtatása: CHR\$(129)

Idézőjel: CHR\$(34)

Skip space (sor formátumának kialakításához): CHR\$(29)

2. FEJEZET

Alapvető algoritmusok

2.1 VEKTOROK

Az egyindexes változót a matematikában megszokott szóhasználatnak megfelelően a számítástechnikában is *vektornak* nevezzük. A két fogalom (a matematikai és számítástechnikai vektorfogalom) nem teljesen azonos jelentésű.

A számítástechnikában a vektor bizonyos típusú és meghatározott számú adat-elemek összessége, amelyekre a programban azonos névvel hivatkozhatunk.

A vektor számára az első hivatkozás előtt DIM utasítással a tárban helyet kell foglalni (l. az 1.10 alfejezetben).

```
10 DIM A(20),A$(20)
20 A(1)=1:A$(1)="ELSO"
30 :
40 :
```

A 10-es programsorban egy 21 elemű (0-tól 20-ig sorszámozott) számokból álló, és egy 21 elemű szövegtartalmú vektort deklaráltunk.

A Commodore 64-esen deklaráció után a numerikus tömb minden elemének értéke nulla, a szöveges tömb minden eleme üres karakter lesz.

A tömbváltozók közül a vektorok kiemelt fontosságúak a programozásban, ezért részletesebben foglalkozunk ezzel a témakörrel.

2.1.1 Vektorok beolvasása

A vektor elemei a program futása közben sokféleképpen kaphatnak értéket. Az értékadás alapvető módja: beolvasás billentyűzetről vagy külső egységről (lemezről, mágnesszalagról).

Ebben a fejezetben a legegyszerűbb esettel foglalkozunk, amikor a gépkezelő az adatokat a billentyűzeten gépeli be.

```
5 REM **** 33.PROGRAM ****
10 REM A VEKTOR BEOLVASASA BILLENTYUZETROL
20 REM 1.VALTOZAT
30 REM A(N)----A FELTOLTENDO VEKTOR
40 REM N-----AZ ELEMEK SZAMA
50 :
60 :
70 :
80 INPUT "AZ ELEMEK SZAMA":N
90 DIM A(N)
100 FOR I=1 TO N
110 PRINT "A( ";I; ")=";
120 INPUT A(I)
130 NEXT I
140 :
```


A 80-as programsorban bekérjük a vektor elemeinek számát, a 90-es sorban deklaráljuk a változót. A képernyőre minden lépésben kiírjuk, hogy hányadik elem beolvasásánál tartunk. Ez a beolvasási módszer nem mindig célravezető. Számos olyan feladat van, amelyben nem tudjuk előre pontosan megadni a vektor méretét.

Ha pl. a gépkezelő az adatokat egy olyan papírról gépeli be, amelyen nincs sor-számozás, akkor a program 80-as sorában feltett kérdésre csak akkor tud válaszolni, ha előre megszámolta az adatlapon levő adatok számát.

Nagyobb adathalmaz esetén ez az előkészítés megkérdőjelezhetné a számítógépes feldolgozás hatékonyságát.

Oldjuk meg a beolvasást az elemek számának pontos ismerete nélkül!

A feladat típusától függően becsüljük meg a vektor méretét! A DIM utasításba felső indexhatárként olyan konstans számértéket kell beírni, amelyet a feldolgozás során sohasem lépünk túl (egyébként ui. a program futása

7 BAD SUBSCRIPT ERROR

hibaüzenettel megszakadna).

A beolvasó ciklus vezérlését az előbbtől eltérően kell megszervezni. Minthogy a beolvasási lépések száma ismeretlen, az adathalmaz végét a program számára egy ún. lezáró vagy elválasztó elemmel (terminátorral) jelöljük.

A lezáró elemet úgy kell megválasztani, hogy az eltérjen a beolvasandó adatoktól. Pl., ha nem negatív számokat olvasunk be, akkor az elválasztó elem lehet egy negatív szám. A megoldás:

```
5 REM **** 34.PROGRAM ****
10 REM A VEKTOR BEOLVASASA BILLENTYUZETROL
20 REM 2.VALTOZAT
30 :
40 :
50 :
60 DIM A(100)
70 PRINT"HAZ ADATBEVITEL VEGET JELEZZE"
80 PRINT"EGY NEGATIV SZAMERTEK BEUTESEVEL!!"
90 PRINT"A( ; I ; )=" ;
100 INPUT A
110 IF A>=0 THEN A(I)=A:I=I+1:GOTO 70
120 PRINT"A TAROLT ADATOK SZAMA:" ; I
130 :
```

A programban kihasználtuk a Commodore 64-es gép néhány sajátosságát. Minthogy ezen a gépen a programok indításakor minden numerikus változó értéke nulla, a beolvasott elemeket számláló változó (I) kezdőértéke is nulla.

Az első beolvasott szám tehát az A vektor nulladik elemébe (A(0)-ba) kerül. Az I változó értékét minden lépésben megnöveljük eggyel, mindaddig amíg a 110-es sorban az $A \geq 0$ reláció teljesül, azaz amíg a program a lezáró elemet (esetünkben negatív számot) nem érzékeli.

A feldolgozandó számokat tehát az

$A(0), A(1), \dots, A(I-1)$

változóban tároltuk, a beolvasott adatok számát beolvasás végén az I változó értéke adja.

Ha a beolvasott adatok tetszőleges valós értéket felvehetnek, a lezáró elem nem lehet numerikus adat.

Módosítsuk a beolvasó programot úgy, hogy lezáró elemként egy különleges karaktert, pl. a "*" karaktert használjuk! Ebben az esetben az adatokat füzérbe kell beolvasnunk, és tárolás előtt (ha az nem * karakter) numerikussá kell konvertálnunk.

```
5 REM **** 35.PROGRAM ****
10 REM A VEKTOR BEOLVASASA BILLENTYUZETROL
20 REM 3.VALTOZAT
30 :
40 :
50 :
60 DIM A(100)
70 PRINT"KEREM GEPELJE BE AZ ADATOKAT!"
80 PRINT"LEZARO KARAKTER: * "
90 PRINT"A( ; I ; )=" ;
100 INPUT A$
110 IF A$<>"*" THEN A(I)=VAL(A$):I=I+1:GOTO 70
120 PRINT"A TAROLT ADATOK SZAMA:" ; I
130 :
```

Alkalmazzuk ugyanezt a megoldást egy neveket tartalmazó karakteres vektor feltöltésére:

```
5 REM **** 36.PROGRAM ****
10 REM A VEKTOR BEOLVASASA BILLENTYUZETROL
20 REM 4.VALTOZAT
30 :
40 :
50 :
60 DIM N$(100):I=1
70 PRINT"KEREM GEPELJE BE AZ ADATOKAT!"
80 PRINT"LEZARO KARAKTER: * "
90 PRINT I ; ".NEV:" ;
100 INPUT A$
110 IF A$<>"*" THEN N$(I)=A$:I=I+1:GOTO 70
120 PRINT"A TAROLT NEVEK SZAMA:" ; I-1
130 :
```

A feltöltést az első elemtől kezdtük. (A 60-as sorban $I=1$). Az A(20) tömb nulladik eleme (A(0)) így üresen maradt, a későbbiekben ezt az elemet tetszőleges adat tárolására felhasználhatjuk.

A példák alapján kiderül, hogy a legegyszerűbb programozási feladat megvalósítására is számtalan lehetőség kínálkozik. Meg kell azonban jegyezni, hogy a bemutatott programrészletek valójában csak programvázlatok. A program helyes működésének alapfeltétele pl. a hibátlan adatbevitel. A bemutatott beolvasó programok egyikébe sem építettünk be adatbeviteli ellenőrzést. Ezekre a kérdésekre még visszatérünk a későbbi fejezetekben.

2.1.2 Vektorok kiírása

Tegyük fel, hogy a számítások eredményét egy vektorban tároltuk. Írjunk egy olyan alprogramot, amely a vektor elemeit kiírja a képernyőre.

A legegyszerűbb megoldás a következő:

```
1000 REM **** 37.PROGRAM ****
1010 REM VEKTOR KIIRASA 1.VALTOZAT
1020 REM N-----A VEKTOR ELEMEINEK SZAMA
1030 REM A(N)--A KIIRANDO VEKTOR
1040 :
1050 :
1060 :
1070 FOR I=1 TO N
1080 PRINT A(I)
1090 NEXT I
1100 RETURN
```

Az alprogramot egyetlen ciklusból építettük fel, és nem ügyeltünk a kiíratási kép megszerkesztésére. Ez az egyszerű megoldás nem mindig célravezető. Egy 100 elemű vektor esetén a számok végigfutnak a képernyőn, és a gépkezelő legfeljebb az utolsó 20 adatot tudja megnézni. Valamivel jobb megoldást kapunk, ha az 1080-as sort megváltoztatjuk:

1080 PRINT(I);

A változó mögé írt pontosvessző hatására a képernyőn a számok egymás mellé kerülnek, de a képernyő tartalma továbbra is áttekinthetetlen marad, és ha megtelt, a felfelé futó számsorokból ismét nem sokat látunk.

Szervezzük meg a kiírást úgy, hogy a képernyősoronként mindig azonos számú (K) elem és a sor elejére a sorban az első elem indexe legyen kiírva! Egy-egy sor kiírása után mindaddig várakozzék a program, amíg a gépkezelő le nem üt egy billentyűt!

A kiíró program:

```
1000 REM **** 38.PROGRAM ****
1010 REM VEKTOR KIIRASA 2.VALTOZAT
1020 REM N-----A VEKTOR ELEMEINEK SZAMA
1030 REM A(N)--A KIIRANDO VEKTOR
1040 REM K-----AZ EGY SORBA IRT ELEMEK SZAMA
1050 :
1060 PRINT "I": I=1
1070 PRINT " I"; TAB(8)
1080 FOR L=1 TO K
1090 PRINT A(I);
1100 I=I+1: IF I>N THEN I=1
1110 NEXT L
1120 PRINT
1130 GETA$: IF A$="" THEN I=1
1140 GOTO 1070
1150 RETURN
```

Az alprogram hívása előtt értéket kell adni az N (elemek száma) és K (egy sorba írt elemek száma) változóknak.

A K változó értékét a számok hosszától függően kell megállapítanunk. Ha a számok tizedesjegyeket tartalmaznak, érdemes kiírás előtt a feladat jellege szerint kerekíteni. Az alábbi programsor beiktatásával a 38-as alprogram két tizedesjegyre kerekítve írja ki a számokat a képernyőre:

1085 A(I)=INT(A(I)*100+0.5)/100

A program végrehajtási idejét jelentősen növelik az indexes változókkal végzett műveletek. Célszerűbb kerekítés előtt a vektor elemét egy numerikus változóba átöltetni (1090-es programsor).

```
1000 REM **** 39.PROGRAM ****
1010 REM VEKTOR KIIRASA 3.VALTOZAT
1020 REM N-----A VEKTOR ELEMEINEK SZAMA
1030 REM A(N)--A KIIRANDO VEKTOR
1040 REM K-----AZ EGY SORBA IRT ELEMEK SZAMA
1050 :
1060 PRINT "I": I=1
1070 PRINT " I"; TAB(8)
1080 FOR L=1 TO K
1090 A=A(I): A=INT(A*100+0.5)/100
1100 PRINT A;
1110 I=I+1: IF I>N THEN I=1
1120 NEXT L
1130 PRINT
1140 GETA$: IF A$="" THEN I=1
1150 GOTO 1070
1160 RETURN
```

Az 1080-astól az 1120-asig terjedő programsorokban kiírtunk egy képernyősor úgy, hogy közben egyesével növeljük a kiírandó elemek indexét, mindaddig, amíg az meg nem haladja az elemek számát (N értéket), ekkor új sort kezdünk.

Fejlesszük tovább a kiíró rutint úgy, hogy az ne álljon meg minden képernyősor kiírása után, hanem csak akkor, ha a teljes képernyőlapot teleírta! Nem kell egymásba ágyazott ciklusokkal dolgozni, ha a sorkezdést egyetlen logikai kifejezés vizsgálatával vezéreljük. Ha egy sorba K elemet szándékozunk írni, akkor az

1, K+1, 2K+1, ...

indexű elemeknél kell sort váltani: vagyis valahányszor a kiírandó elem indexe K-val elosztva 1 maradékot ad. Ha az index I, akkor a K-val való osztás maradékát az

$I - \text{INT}(I/K) * K$

aritmetikai kifejezés adja. Értéke alapján vagy az adott, vagy pedig az új sorban folytatjuk a kiírást. A lapváltást a kiírt sorok számlálásával figyelhetjük.

A 40-es alprogramban az SI változó jelzi a kiírt sorok aktuális számát. Valahányszor értéke meghaladja az egy képernyőre tervezett sorok számát, meghívunk egy „lapváltó” rutint, amely tetszőleges ideig várakozik, majd letörli a képernyőt, így a kiírás üres lapon folytatódhat.

A lapra kiírt sorok számát jelző változó (SI) értékét ekkor ismét nullára állítjuk:

```
1000 REM ****40.PROGRAM****
1010 REM VEKTOR KIIRASA 4.VALTOZAT
1020 REM N-----A VEKTOR ELEMEINEK SZAMA
1030 REM A(N)---A KIIRANDO VEKTOR
1040 REM S-----A SOROK SZAMA EGY LAPON
1050 REM K-----AZ ADATOK SZAMA EGY SORBAN
1060 :
1070 SI=0:PRINT" "
1080 FOR I=1 TO N
1090 IF I=INT(I/K)*K<>1 THEN 1120
1100 IF SI>S THEN SI=0:GOSUB 1170
1110 PRINT:SI=SI+1
1120 A=INT(A(I)*100+0.5)/100
1130 PRINTA;
1140 NEXTI
1150 RETURN
1160 :
1170 REM LAPVALTAS *****
1180 PRINT:PRINT"BARMELY BILLENTYURE TOVABB!"
1190 GETA$:IFA$="" THEN 1190
1200 PRINT" "
1210 RETURN
```

A kiíratási kép az utolsó megoldással ugyan biztosan áttekinthetőbb, de egyáltalán nem biztos, hogy esztétikus is lesz.

Ha az egyes számok eltérő számú számjegyet tartalmaznak, akkor a tizedesponthoz nem kerülnek egymás alá. Az sem mindegy, hogy a kiírást képernyőre vagy a nyomtatóra szervezzük, hiszen a képernyőn 40, míg a nyomtatón általában 60—80 karakter fér el egy sorban.

Az esztétikus képernyő-, ill. nyomtatási képek megszerkesztése géptől-, ill. programnyelvektől függő programozástechnikai eszközöket igényel.

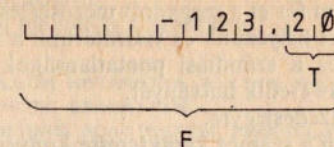
A BASIC nyelv sajnos nem tartalmaz a numerikus változók formázott kiírására szolgáló utasításokat. Arra ugyan van lehetőség, hogy a kiírást mindig a képernyő egy adott pozíciójától kezdjük (TAB utasítás), de ezzel a tizedesponthoz még nem kerülnek egymás alá.

```
123.1
-1.243
0.25
.
```

A számítások eredményét a felhasználó általában nem a fenti, hanem az alábbi formában várja:

```
123.100
-1.243
0.250
```

A kiírást tehát a tizedesponthoz kell igazítanunk. A BASIC-ben ez csak úgy lehetséges, ha kiírás előtt a számokat karakteres változóként kezelve előre rögzített hosszúságra egészítjük ki, előlről megfelelő számú szóköz, hátulról pedig megfelelő számú 0 karakterrel. Jelöljük a kiírás formátumát F.T-vel, ahol F a szám teljes hossza, T pedig a tizedesjegyek száma. A számok átalakítás utáni alakjára láthatunk példát az 1. ábrán.



1. ábra Formázott kiírás

A számok formázását végző 41-es alprogram nem egyéb, mint a Commodore 64-es személyi számítógép és a BASIC programnyelv nyelvének bizonyítéka, és egyben utalás arra, hogy aki komolyabban szeretne foglalkozni a számítástechnikával, feltétlenül meg kell ismerkednie a fejlettebb programnyelvekkel, mint pl. PASCAL, FORTRAN, PL1 stb.

A formázó rutin leírása

Hívása előtt értéket kell adni a következő változóknak:

A — a formázandó szám;

F — a mezőszélesség;

T — a tizedesjegyek száma;

N\$, C\$, U\$ — "0", " ", " ", ill. szóköz karakterekkel feltöltött segédváltozók.

A számot formázott alakban az A\$ változóban kapjuk vissza. A mezőszélességet (F) az adatok nagyságrendje alapján kell meghatározni. Féllogaritmikus alakú számok esetében (pl. 1.3 E-12) értéke legalább hattal legyen nagyobb a tizedesjegyek számánál.

```
3000 REM **** 41.PROGRAM ****
3010 REM SZAMOK FORMAZASA
3020 M=F-T-2
3030 E$=" ":E=SGN(A):A=ABS(A)
3040 IFE=-1 THEN E$="-"
3050 IF M<1 OR INT(A)>10*M-1 THEN A$=LEFT$(C$,F):RETURN
3060 KT=1:FOR Y=1 TO T:KT=KT*10:NEXT Y
3070 A=INT(A*KT+.5)/KT
3080 A$=STR$(A):L=LEN(A$)-1:A$=RIGHT$(A$,L)
3090 IFL<4 THEN 3110
3100 IF MID$(A$,L-3,1)="E" THEN A$=LEFT$(U$,F-L-1)+E$+A$:RETURN
3110 IF LEFT$(A$,1)=". " THEN A$="0"+A$:L=L+1
3120 FOR U=1 TO L
3130 IF MID$(A$,U,1)=". " THEN 3170
3140 NEXT U
3150 IFT=0 THEN A$=LEFT$(U$,M-L+1)+E$+A$:RETURN
3160 A$=LEFT$(U$,M-L)+E$+A$+"."+LEFT$(N$,T):RETURN
3170 B$=LEFT$(U$,M-U+1)+E$+LEFT$(A$,U-1)
3180 D$=RIGHT$(A$,L-U)+LEFT$(N$,T-L+U)
3190 A$=B$+"."+D$
3200 RETURN
```


A rutin felépítése

- 3020 Meghatározzuk a tizedespont előtti számjegyek számát (két hely szükséges a tizedespont és az előjel számára).
- 3030—3040 Leválasztjuk a szám előjelét, és az E változóban tároljuk (pozitív szám esetén szóköz karaktert).
- 3050 Ha a szám nem fér el a megadott mezőszélességen, az A-ba F számú "*" karaktert helyezünk és visszatérünk a főprogramba.
- 3060 A kerekítéshez a számítási pontatlanságok elkerülésére szorzással számítjuk ki 10T-edik hatványát.
- 3070 Kerekítés T tizedesjegyre.
- 3080 Kerekítés után a számot karakteressé konvertáljuk, és leválasztjuk a szóköz karaktert, amelyet az STR\$ függvény helyez a szám elé. L: a karakterlánc hossza.
- 3090—3100 Megvizsgáljuk, hogy a szám féllogaritmikus alakú-e (l. a 1.2 pontot). Az ilyen alakú számokat balról megfelelő számú szóközzel egészítjük ki, és visszatérünk a főprogramba.
- 3110 Az 1-nél kisebb számok elé 0 karaktert írunk.
- 2120—3140 Megkeressük a karakterláncban a tizedespontot.
- 3150—3200 Kiegészítjük a számot előlről megfelelő számú szóköz-, hátulról nulla karakterrel, majd visszatérünk a főprogramba.

Összefoglalásként bemutatunk egy olyan programot, amely a tárgyalt alprogramok-ra építve kiírja képernyőre egy adott függvény adott intervallumban vett értéktáblázatát.

Sajnos a program elkészítése közben most is meg kell alkudnunk a BASIC nyelv adta lehetőségekkel.

A függvény táblázatát kiíró program akkor lehetne általános, ha a tabellázandó függvény egyenlete is a bemenő adatok között szerepelhetne. A C 64-es BASIC nyelve sajnos ezt nem teszi lehetővé. A gép belső felépítésének, bizonyos tárcímek szerepének ismeretében ugyan a feladat megoldható, de erre a könyvnek ebben a részében szándékosan nem térünk ki.

A tetszőleges függvény, tetszőleges intervallumbeli tabellázása BASIC nyelven ui. további programozási gondokhoz vezet. Ha a függvény az intervallum valamely osztópontjában nincs értelmezve, akkor a program futása a meg nem engedett aritmetikai művelet hatására hibaüzenettel megszakad.

A jó programmal szemben támasztott alapvető követelmény, hogy a felhasználó ne adhasson meg (szándékosan sem és véletlenül sem) olyan adatokat, amelyek feloldozására a program nincs felkészítve, amelyek a program további tevékenységét határozatlanná teszik, vagy hibaüzenettel kísért programmegszakítást okoznak.

Hibaüzenet alatt itt nem a program által hanem az operációs rendszer által kiírt hibaüzenetet kell érteni. Nézzünk erre egy egyszerű példát!

```
10 INPUT "AZ INTERVALLUM HATARA1(A,B):" ; A,B
20 INPUT "AZ OSZTOPONTOK SZAMA:" ; N
30 DX=(B-A)/N
READY.
```

Futtassuk le a bemutatott programrészletet, és a program által feltett kérdésre a számadat beütése helyett válaszoljunk a RETURN billentyű leütésével.

A következő felirat a képernyőn:

? DIVISION BY ZERO IN 30

A program használatától nem várhatjuk el, hogy az operációs rendszer angol nyelvű hibaüzenetét értelmezze, majd a kérdéses (30-as) programsor elemzése után eldöntse, hogy mit csinált rosszul.

A helyes programozási megoldás a következő:

```
10 INPUT "AZ INTERVALLUM HATARA1(A,B):" ; A,B
20 INPUT "AZ OSZTOPONTOK SZAMA:" ; N
30 IF N<=0 THEN PRINT "HIBAS ADAT! KEREM ISMETELJE MEG!" ; GOTO 20
40 DX=(B-A)/N
```

A program kiszűri a hibás adatot, és arról tájékoztatja a gépkezelőt.

Térjünk vissza a tetszőleges függvényt tabellázó program tervezéséhez!

Ahhoz, hogy a programot tetszőleges aritmetikai vagy egyéb hiba esetére fel tudjuk készíteni, egy általános hibakezelő utasításra lenne szükségünk.

Ilyen utasítás a PASCAL, a PL1, ill. némely BASIC változatban ismert:

```
ON ERROR GOTO ...
ill. ON ERROR GOSUB ...
```

Az utasítás hatására bármilyen, egyébként hibaüzenettel kísért, programmegszakítást okozó hiba esetén a program végrehajtását az általunk meghatározott soron vagy alprogramban folytathatjuk.

Egyes programnyelvekben a hiba típusát is lekérdezhetjük (aritmetikai hiba, index-határ túllépése stb.).

Ha a függvényértékek kiszámítása közben aritmetikai hiba lép fel, akkor a hibakezelő rutinban az adott függvényérték helyére pl. tájékoztatásul egy különleges karaktert vagy karaktersorozatot írhatunk.

A következő programot a BASIC nyelv által megszabott lehetőségek szerint írtuk meg. A formázó rutint nem nyomtattuk ki, mivel azt már korábban bemutattuk (l. 41. program).

```
5 REM **** 42. PROGRAM ****
10 U$=""
20 DEF FN F(X)=SIN(X)
30 PRINT "TÁBLÁZAT A NYOMTATORA--N"
40 PRINT "KEPERNYORE--K"
50 GET G$: IF G$="" THEN 50
60 IF G$("<"N" AND G$("<"K" THEN 50
70 IF G$="N" THEN P=4:SR=60:GOTO 90
80 P=3:SR=40
90 OPEN I,P
100 INPUT "AZ INTERVALLUM ALSO HATARA" ; X1
110 INPUT "AZ INTERVALLUM FELSO HATARA" ; X2
120 INPUT "LEPESKOZ" ; DX
130 IF X2<X1 OR DX<=0 THEN 100
140 N=INT((X2-X1)/DX)+1:DIM A(N):X=X1
150 FOR I=1 TO N
160 A(I)=FN F(X):X=X+DX
170 NEXT I
180 PRINT "A KIIRAS FORMATUMA"
```



```

190 INPUT"MEZOSZELESSEG (FEJLEC, TABLAZAT)";FF,FT
200 INPUT"TIZEDESJEGYEK (FEJLEC, TABLAZAT)";TF,TT
210 S=INT(SR/FT)-2
220 GOSUB1000
230 PRINT#1:CLOSE1
240 STOP
250 :
1000 REM A VEKTOR ELEMEINEK KIIRASA A KEPERNYORE
1010 REM *****
1020 REM 4.VALTOZAT
1030 REM N-----A VEKTOR ELEMEINEK SZAMA
1040 REM A(N)--A KIIRANDO VEKTOR
1050 X=X1:SI=0:GOSUB2050
1060 FOR I=1 TO N
1070 IF I-INT(I/S)*S<>1 THEN 1110
1080 IF SI>15 AND P=3 THEN SI=0:GOSUB 2000
1090 A=X:F=FF:T=TF:GOSUB3000:X=X+S*DX:SI=SI+1:F=FT:T=TT
1100 PRINT#1:PRINT#1,A$;"I ";
1110 A=A(I):GOSUB 3000
1120 PRINT#1,A$;
1130 NEXT I
1140 RETURN
1150 :
2000 REM LAPVALTAS
2010 REM *****
2030 PRINT:PRINT"BARMELY BILLENTYURE TOVABB!"
2040 GET G$:IF G$="" THEN 2040
2050 PRINT#1:PRINT#1,"INTERVALLUM: [";X1;" ";X2;" ] "
2060 FOR U=1 TO 40
2070 PRINT#1,"=";
2080 NEXT:PRINT#1," X"+LEFT$(U$,FF-2);
2090 FORZ=0TOS-1:A=DX*Z:F=FT:T=TT:GOSUB3000:PRINT#1,A$;
2100 NEXTZ:PRINT#1
2110 FOR U=1 TO 40
2120 PRINT#1,"=";
2130 NEXT U
2140 RETURN
2150 :

```

A program felépítése

Sorszám	Megnevezés	Feladat	Változók és eredmények	
			INPUT	OUTPUT
240	Főprogram	Segédváltozók feltöltése. A fgv. meghatározása. Adatbeolvasás, deklarációk. A fgv.-értékek kiszámítása és tárolása az A vektorban. A kiíró rutin hívása.	X1, X2 határok; DX lépésköz; FF, FT mezőszélesség; TF, TT a tizedesjegyek száma	A a fgv.-értékeket tartalmazó vektor; S az egysorba írandó adatok száma; N az elemek száma
1000 1140	Kiíró rutin	A táblázat kiírása képernyőre laponként. A lapváltó és a formázó rutinok hívása.	X1, DX, N, S, A(N), A\$	A képernyőn megjelenő táblázat.
2000 2140	Lapváltó rutin	A fejléc kiírása és várakozás egy billentyű leütésére.		A fejléc
3000	Formázó rutin	A szám átalakítása rögzített hosszúságúra, rögzített számú tizedesjeggyel.	F mezőszélesség; T a tizedesjegyek száma; A az átalakítandó szám	A\$ a formázott szám.

2.1.3 Műveletek vektorokkal

A vektorok feldolgozásának algoritmusai igényességüket és bonyolultsági fokukat tekintve igen széles terjedelműek. A gyorsrendező algoritmusokkal pl. kötetnyi irodalom foglalkozik.

Az alapműveletek (mint pl. két vektor összege) nagyon egyszerűen programozhatóak, de itt is könnyen találunk példát arra, hogy a programozás a matematikától eltérő gondolkodásmódot igényel, és a program elkészítése előtt érdemes alaposan végiggondolni, hogy milyen adatokból milyen eredményt szeretnénk kapni.

Írjunk pl. programrészletet, amely egy vektor elemeit elosztja a vektor első elemével!

```

1000 REM **** 43.PROGRAM ****
1010 REM HOGYAN NEM LEHET??
1020 :
1030 FOR I=1 TO N
1040 A(I)=A(I)/A(1)
1050 NEXT I
1060 :
1070 :
1080 :

```


Próbáljuk meg gép nélkül eldönteni, hogy a programrészlet futtatása valóban a kívánt eredményt hozza-e!

Tegyük fel, hogy a vektor elemei:

$$A(1)=2, \quad A(2)=3, \quad A(3)=4.$$

Kövessük a program működését soronként úgy, hogy közben a változók aktuális tartalmát beírjuk egy táblázatba:

I	A(1)	A(2)	A(3)
	2	3	4
1	1	3	4
2	1	3	4
3	1	3	4

A helyes eredmény az $A(1)=1$, $A(2)=1.5$, $A(3)=2$ lett volna, de a programrészlet képzeletbeli futtatásából nem ezt kaptuk.

Milyen hibát követtünk el? Az első cikluslépésben végrehajtott művelet:

$$A(1)=A(1)/A(1)$$

Az eredmény $A(1)=1$, így a többi elemet nem az eredeti $A(1)$ értékkel, hanem 1-gyel osztjuk végig. A helyes megoldás:

```
1000 REM **** 44.PROGRAM ****
1010 :
1020 :
1030 X=A(1)
1040 FOR I=1 TO N
1050 A(I)=A(I)/X
1060 NEXT I
1070 :
1080 :
1090 :
```

A következőkben a leggyakoribb és legalapvetőbb vektorműveletek programjaival foglalkozunk.

Határozzuk meg az $a=[a_1, a_2, \dots, a_n]$
és $b=[b_1, b_2, \dots, b_n]$

vektorok összegvektorát, melynek elemei:

$$c_i = a_i + b_i \quad (i=1, \dots, n).$$

Tekintve, hogy az eredmény is vektor, nem szabad megfeledkezni az összegvektor deklarálásáról.

```
10 REM **** 45.PROGRAM ****
20 :
30 REM KET VEKTOR OSSZEGE
40 :
50 DIM A(N), B(N), C(N)
60 :
200 FOR I=1 TO N
210 C(I)=A(I)+B(I)
220 NEXT I
230 :
```

A program elkészítésénél feltételeztük, hogy az N változó (a vektorok elemeinek száma) az 50-es programsor végrehajtása előtt értéket kapott.

A matematikából ismert fogalom két vektor skaláris szorzata. Az $a=[a_1, a_2, \dots, a_n]$ és $b=[b_1, b_2, \dots, b_n]$ vektorok skaláris szorzata:

$$C = \sum_{i=1}^n a_i b_i$$

A skaláris szorzat sokféle gazdasági tartalommal ruházható fel. Tegyük fel pl., hogy egy raktári készlet összértékét határozzuk meg a raktáron levő anyagok egységára és mennyisége ismeretében. Az N anyagfélése egységárait és mennyiségét az A és B vektorokban tároltuk. A raktári készletet a következő alprogram számítja ki:

```
1000 REM **** 46.PROGRAM ****
1010 REM VEKTOROK SKALARIS SZORZATA
1020 REM A(N), B(N) AZ OSSZESZORZANDO VEKTOROK
1030 REM C ----- A SKALARIS SZORZAT
1040 :
1050 :
1060 :
1070 C=0
1080 FOR I=1 TO N
1090 C=C+A(I)*B(I)
1100 NEXT I
1110 RETURN
```

Az alprogramot természetesen csak úgy hívhatjuk meg, ha előzetesen a tömböket deklaráltuk és feltöltöttük adatokkal.

A skaláris szorzat értékét a C változóban kapjuk vissza.

2.2 SZÁMLÁLÁS, OSZTÁLYBA SOROLÁS

Igen gyakori programozási feladat, hogy egy vektorváltozó elemei közül ki kell választani azokat, amelyek bizonyos feltételeknek eleget tesznek. Nevezzük ezeket az elemeket a továbbiakban kitüntetett elemeknek. A legegyszerűbb eset, amikor a kitüntetett elemeknek csak a számát kell meghatározni. Az ilyen jellegű feladatokat számlálási feladatoknak nevezzük. A számlálási feladatok programozástechnikailag azonos szerkezetűek. A vektor elemeit rendre megvizsgálva, egy feltétel alapján eldöntjük, hogy az kitüntetett elem vagy sem.

Ha igen, egy számláló értékét eggyel megnöveljük, egyébként folytatjuk az eljárást a következő elemnél.

Határozzuk meg egy számokat tartalmazó vektorváltozó azon elemeinek számát, amelyek egy adott intervallumba esnek! A programrészletet írjuk meg alprogramként, amelyet a vektorváltozó deklarálása és beolvasása után hívhatunk meg!

```
1000 REM **** 47.PROGRAM ****
1010 REM VALOGATAS
1020 REM A(N)--A VEKTOR
1030 REM A-----AZ INTERVALLUM KEZDOPONTJA
1040 REM B-----AZ INTERVALLUM VEGPONTJA
1050 REM K-----AZ INTERVALLUMBA ESO ELEMEK SZAMA
1060 :
1070 :
1080 :
1090 K=0
1100 FOR I=1 TO N
1110 IF A(I)>A AND A(I)<B THEN K=K+1
1120 NEXT I
1130 RETURN
```

Az alprogram hívására példaként írtunk egy olyan programot, amely N számú véletlen szám közül meghatározza a (0.1; 0.2) intervallumba esők számát.

Ha az RND(0) függvény valóban egyenletes eloszlású számokat generál, akkor nagy N érték esetén eredményül N/10-hez közel eső számértéket kell kapnunk.

```
5 REM **** 48.PROGRAM ****
10 INPUT "A VELETLEN ERTEKEK SZAMA";N
20 DIM A(N)
30 FOR I=1 TO N
40 A(I)=RND(0)
50 NEXT I
60 A=0.1:B=0.2
70 GOSUB 1000
80 PRINT "AZ ";N;" SZAM KOZUL "
90 PRINT "AZ ";A;"-" ;B;" INTERVALLUMBA ESOK SZAMA:";K
100 STOP
```

A 47-es alprogramot kis módosítással átalakíthatjuk úgy, hogy a kitüntetett elemeknek ne csak a számát, hanem az átlagát is kiszámítsa. A szükséges kiegészítések:

```
1000 REM **** 49.PROGRAM ****
1010 REM VALOGATAS,ATLAG
1020 REM A(N)--A VEKTOR
1030 REM A-----AZ INTERVALLUM KEZDOPONTJA
1040 REM B-----AZ INTERVALLUM VEGPONTJA
1050 REM K-----AZ INTERVALLUMBA ESO ELEMEK SZAMA
1060 REM S-----AZ INTERVALLUMBA ESO ELEMEK ATLAGA
1070 :
1080 :
1090 K=0:S=0
1100 FOR I=1 TO N
1110 IF A(I)>A AND A(I)<B THEN K=K+1:S=S+A(I)
1120 NEXT I
1130 IF K=0 THEN RETURN
1140 S=S/K
1150 RETURN
```

Számlálási feladatra vezethető vissza egy adathalmaz elemeinek *osztályba sorolása*. Tegyük fel, hogy az a_i ($i=1, \dots, n$) számokat k számú osztályba kell besorolnunk, és meg kell határozni az egyes osztályokba eső elemek számát, ill. relatív gyakoriságát. Ha a j -edik osztályba eső elemek száma k_j , akkor a relatív gyakoriság:

$$r_j = \frac{k_j}{n}.$$

Jelölje m az adathalmaz minimális, M pedig a maximális elemét. Akkor az egyenközű osztályok terjedelme:

$$h = \frac{M-m}{k}.$$

Az osztályba soroláshoz először meg kell határozni az adathalmaz minimális és maximális elemét, majd az osztályok méretét. A következő alprogram kiválasztja a vektor minimális és maximális elemét.

```
1000 REM **** 50.PROGRAM ****
1010 REM A VEKTOR MINIMALIS ES MAXIMALIS ELEMENEK MEGHATAROZASA
1020 REM A(N)----A VEKTOR
1030 REM MA-----A MAXIMALIS ELEM
1040 REM MI-----A MINIMALIS ELEM
1050 :
1060 :
1070 MA=A(1):MI=A(1)
1080 FOR I=2 TO N
1090 IF A(I)<MI THEN MI=A(I):GOTO 1110
1100 IF A(I)>MA THEN MA=A(I)
1110 NEXT I
1120 RETURN
```

Az alprogram hívása után az osztályok mérete a

$$H = (MA - MI)/K$$

értékadással számítható ki.

Az 1070-es sorban az adathalmaz első elemét jelöltük meg minimális és maximális elemként egyaránt.

Az 1080-astól az 1110-esig terjedő ciklusban lépésenként megvizsgáljuk, hogy az adathalmaz következő eleme kisebb-e, mint az addigi legkisebb elem (MI), ill. nagyobb-e, mint az addigi legnagyobb elem (MA).

A feltételek teljesülésétől függően az addig legkisebb, ill. legnagyobb elem tartalmát felülírjuk, vagy változatlanul hagyjuk és folytatjuk az eljárást.

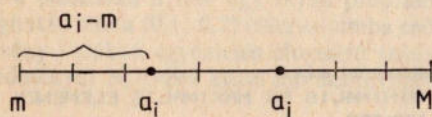
Ismerve az osztályok méretét, most meghatározhatjuk a relatív gyakoriságokat. Az osztálybasoroló rutin hívásakor nem szabad megfeledkezni arról, hogy eredményül annyi relatív gyakoriságot kapunk, ahány osztályt képeztünk, tehát ezek számára egy újabb vektorváltozót kell deklarálni.


```

2000 REM **** 51.PROGRAM ****
2010 REM OSZTALYBA SOROLAS EGYENKOZU OSZTOPONTOKKAL
2020 REM A(N)---AZ OSZTALYBA SORLANDO ADATOK
2030 REM K-----AZ OSZTALYOK SZAMA
2040 REM R(K)---A RELATIV GYAKORISAGOK
2050 IF H=0 THEN K=1:R(1)=1:RETURN
2060 FOR I=1 TO N
2070 J=INT((A(I)-MI)/H)+1
2080 IF J>K THEN J=K
2090 R(J)=R(J)+1
2100 NEXT I
2110 FOR I=1 TO K
2120 R(I)=R(I)/N
2130 NEXT I
2140 RETURN

```

Az osztályba sorolást nagyon sokféleképpen lehet programozni. A bemutatott rutin elve (l. 2. ábra):



2. ábra Osztályba sorolás egyenközű felosztással

Az a_i elem osztálysorszámát megkapjuk, ha megvizsgáljuk, hogy az $a_i - m$ különbségre (ahol m a minimális elem) hányszor lehet felmérni az osztály terjedelmét, és a kapott értéket 1-gyel megnöveljük.

Az intervallumok határpontjára eső elemeket így mindig abba az osztályba soroljuk, amelyeknek ez a pont a kezdőpontja. Az ábra alapján pl.:

$$\left\lceil \frac{a_j - m}{h} \right\rceil = 4 \left\lceil \frac{a_i - m}{h} \right\rceil = 2$$

Az a_j elem tehát az ötödik, az a_i pedig a harmadik osztályba kerül, ahol h az osztályok méretét, m a minimális elemet, a_j és a_i pedig az i -dik, ill. a j -dik elemet jelöli.

A határpontra eső elemeket a feladat jellegétől függően az általuk határolt osztályok bármelyikébe (de csak az egyikbe) soroljuk. Bármelyik megoldás mellett döntünk, a minimális és maximális (ill. az ezekkel egyenlő) elemek valamelyike kimarad az osztályozásból. A határelemeket az általuk felülről határolt osztályba sorolva a minimális elem a nulladik osztályba kerülne.

Az általunk követett megoldásban viszont a maximális, ill. az ezzel egyenlő elemeket a $K+1$ -edik osztályba kellene sorolni, ami éppúgy nem létezik, mint a nulladik osztály. Ezt az esetet az alprogram 2060-as sorában vizsgáljuk, ahol az utolsó intervallum végpontjára eső elemeket önkényesen az utolsó (K -adik) osztályba soroljuk.

Az adathalmazok statisztikai vizsgálatának, mérési eredmények kiértékelésének kiindulópontja általában a minimális-, maximális elem kiválasztása, az átlag meghatározása, az osztályba sorolás. Az elkészült alprogramokból felépíthetünk egy olyan programot, amely feltárja az adathalmaz szerkezetének alapvető tulajdonságait.

Szemléltetésként egy tankört az évfolyam reprezentatív részhalmazának tekintve, statisztikát készítünk a hallgatók testmagasságáról. A programot kiegészítettük az előbbieken kívül egy olyan alprogrammal, amely a relatív gyakoriságok alapján oszlopdiagrammal ábrázolja az adatok egyes osztályok közötti megoszlását.

A grafikus ábrát készítő alprogramban előfordul néhány olyan programozástechnikai megoldás, amely kifejezetten a Commodore 64-es gép sajátosságaiból ered, ezért ezt az alprogramot részletesen dokumentáljuk.

Az oszlopokat a kurzorral azonos méretű négyzetekből építjük fel, így a képernyőn esztétikusan maximum 10 osztályt tudunk szemléltetni.

```

3000 REM **** 52.PROGRAM ****
3010 REM OSZLOPDIAGRAM(MAX. 12 OSZLOP)
3020 FOR I=1 TO K:A(I)=R(I):NEXT I:N=K
3030 GOSUB 1000
3040 E=20/MA
3050 IF K<=6 THEN SZ=3:GOTO 3070
3060 SZ=2
3070 KP=(40-K*(SZ+1))/2
3080 POKE 781,22:POKE 782,4:SYS 58636
3090 FOR I=1 TO 35:PRINT " ";:NEXT I
3100 FOR I=2 TO 22:POKE 781,I:POKE 782,4:SYS 58636:PRINT "I":NEXT I
3110 S=2
3120 FOR I=100*MA TO 10 STEP -10
3130 POKE 781,S:POKE 782,0:SYS 58636
3140 PRINT I:S=S+E*0.1
3150 NEXT I
3160 FOR I=1 TO K
3170 C=INT(RND(0)*15)+1:IF C=6 THEN 3170
3180 POKE 646,C
3190 M=R(I)*E
3200 IF M=0 THEN 3250
3210 FOR L=1 TO M
3220 POKE 781,22-L:POKE 782,KP:SYS 58636
3230 FOR T=1 TO SZ:PRINT "█":NEXT T
3240 NEXT L
3250 KP=KP+SZ+1
3260 NEXT I
3270 POKE 781,23:POKE 782,4:SYS 58636
3280 PRINT "MAX. GYAK: ";MA;" MIN. GYAK: ";MI
3290 GETA$:IFA$="" THEN 3290
3300 RETURN

```

Az alprogramban használt változók:

- R(I) — a relatív gyakoriságokat tartalmazó vektor;
- K — az osztályok száma;
- MA — a legnagyobb relatív gyakoriság;
- SZ — az oszlopok karakterszélessége;
- E — egy egység ábrázolására szolgáló karakterek száma;
- C — az oszlop színe;
- M — az oszlop magassága;
- KP — az oszlop kezdőpozíciója a képernyőn.

- 3020—3030 A legnagyobb és legkisebb relatív gyakoriság meghatározása. A 1000-es szubrutin kiválasztja az A vektorváltozó maximális és minimális elemét. Mivel mi most az R vektor legnagyobb elemét szeretnénk meghatározni, az 1000-es szubrutin hívása előtt az R vektort áttöltjük az A vektorba. Igaz, hogy ezzel az A vektor eredeti tartalmát elrontjuk, de az eredeti adathalmazra a program további részében már nincs szükségünk.
- 3110—3150 Megrajzoljuk a függőleges beosztást. A legfelső osztópont a maximális relatív gyakoriság értékét jelzi százalékosan. Ezt a képernyő második sorába írjuk ($S=2$), a többi osztópont a 0.1 értéknek (10%-nak) megfelelő számú karakterrel lejjebb kerül.
- 3160—3260 Az oszlopok kirajzolása. A rajzolás három egymásba ágyazott ciklussal szerveztük meg. A legbelső ciklus (3230-as sor) megrajzolja egy adott oszlop egyetlen sorát. Az oszlop szélességét az SZ-ben tárolt számítás (2 vagy 3) szabja meg. A középső ciklus (3210-estől 3240-esig) egy oszlopot rajzol. Az oszlop magasságát az M változó tartalmazza. Ha $M=0$, ezt a lépést kihagyjuk. M értékét az E egységnek megfelelően az R(I) relatív gyakorisággal arányosan számítjuk ki (3190-es). Az aktuális oszlop kezdetét a KP változó tartalmazza, amelynek értékét minden oszlop kirajzolása után az oszlop szélességével, ill. az üresen hagyott egy karakter szélességgel megnöveljük (3250-es). Az oszlop színét véletlenszám-generátorral választjuk ki a 3170-es sorban, a kapott 0 és 15 közé eső színkódot beírjuk a 646-os tárcímre, amely mindig az aktuális karakter színét tartalmazza.
- 3040 A grafikus ábrázoláshoz a képernyőn maximálisan 20 sort tudunk felhasználni (a maradék néhány sor szükséges a szöveg kiírására, a vízszintes tengely megrajzolására). A legnagyobb gyakoriságot tehát 20 karakter magasságú oszloppal ábrázolhatjuk. Ennek megfelelően arányosan kell kiszámítani a többi oszlop magasságát. A méret alapegységét az E tartalmazza.
- 3050—3060 Attól függően, hogy hatnál több vagy kevesebb osztályt ábrázolunk, az oszlopokat 3, ill. 2 karakter szélességben rajzoljuk meg.
- 3070 Az ábrát a képernyő közepére kell elhelyezni. Az első oszlop kezdőpozícióját (KP) ismét az oszlopok száma és szélessége határozza meg.
- 3080 Minthogy ezen a gépen nincs olyan BASIC utasítás, amellyel a kurzort a képernyő sor-oszlop koordinátájú pontjába mozgathatnánk, erre a célra az 58636-os címen kezdődő rendszerrutint használjuk. A pozícionáló rutint SYS 58636 BASIC utasítással hívjuk meg. Hívása előtt a 781-es és a 782-es tárcímekre be kell írni a kívánt sor-, ill. oszlopkoordinátát.
- 3090 Megrajzoljuk a vízszintes tengelyt. A legkülső ciklust az I ciklusváltozó vezérli, 1-től K-ig, ami az oszlopok (osztályok) száma.
- 3270—3280 A képernyő legalsó sorába kiírjuk a maximális és minimális relatív gyakoriságot. Végül, hogy az ábrát a gépkezelő tetszőleges ideig szemlélhesse, addig várakozik a program, amíg le nem üt egy billentyűt.
- Miután az egyes részfeladatokat kellő alaposítással megoldottuk, a főprogram elkészítése igen egyszerűvé vált. Be kell olvasni az adatokat, az egyes alprogramokat

megfelelő sorrendben meg kell hívni, ill. a részeredményeket a képernyőre ki kell írni.

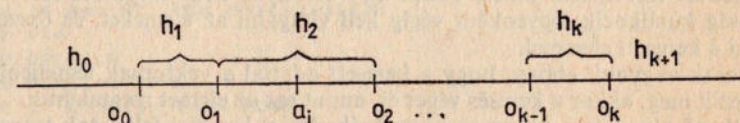
```

5 REM **** 53.PROGRAM ****
10 INPUT "A CSOPORT LETSZAMA";N
20 DIM A(N)
30 PRINT "KEREM A CSOPORT TAGJAINAK TESTMAGASSAGAIT"
40 S=0
50 FOR I=1 TO N
60 PRINT I; ".HALLGATO MAGASSAGA";
70 INPUT A(I)
80 S=S+A(I);REM OSSZEGZES AZ ATLAGSZAMITASHOZ
90 NEXT I
100 PRINT "ATLAGOS TESTMAGASSAG:";S/N
110 GOSUB 1000:REM MIN-MAX*****
120 PRINT "A MAXIMALIS TESTMAGASSAG:";MA
130 PRINT "A MINIMALIS TESTMAGASSAG:";MI
140 INPUT "OSZTALYOK SZAMA";K
150 DIM R(K)
160 H=(MA-MI)/K
170 GOSUB 2000:REM OSZTALYBA SOROLAS*****
180 PRINT "AZ OSZTALYBA SOROLAS EREDMENYE"
190 FOR J=1 TO 30:PRINT " ";:NEXT J:PRINT
200 FOR I=1 TO K
210 PRINT I; "OSZTALY:";R(I)
220 NEXT I
230 PRINT "BARMELY BILLENTYURE TOVABB!!"
240 GETAS:IF AS="" THEN 240
250 PRINT " "
260 GOSUB 3000:REM OSZLOPDIAGRAM*****
270 STOP

```

Az elemek osztályba sorolására a leírt módszer csak akkor alkalmas, ha a felosztás egyenközi.

Ellenkező esetben egy vektorváltozóban meg kell adni az osztályhatárokat.



3. ábra Osztályba sorolás változó méretű osztályokkal

A 3. ábrán $o_0, o_1, o_2, \dots, o_k$ jelöli az osztópontokat. Az osztópontok által határolt véges osztályok száma k . Jelölésük: $h_1, h_2, \dots, h_k$.

Ha az adathalmaz elemeinek nagyságrendjét nem ismerjük, akkor a valódi osztályokon kívül fel kell venni még két osztályt.

Egyet a legalsó határ alá (h_0), és egyet a felső határ fölé eső elemek számára (h_{k+1}). A programrészlet szerkezete eltér az előzőtől.


```

2000 REM **** 54.PROGRAM ****
2010 REM VALTOZO HUSSZUSAGU OSZTALYKOZOKKEL
2020 REM A(N)---AZ ADATVEKTOR
2030 REM K-----A VEGES OSZTALYOK SZAMA
2040 REM O(K)---AZ OSZTOPONTOKAT TARTALMAZO VEKTOR
2050 REM R(K+1)-A RELATIV GYAKORISAGOK
2060 :
2070 :
2080 :
2090 FOR I=1 TO N
2100 J1=0
2110 FOR J=0 TO K
2120 IF A(I)<=O(J) THEN R(J)=R(J)+1:J1=1:GOTO2150
2130 NEXT J
2140 IF J1=0 THEN R(K+1)=R(K+1)+1:PRINT"N"
2150 NEXT I
2160 FOR J=0 TO K+1
2170 R(J)=R(J)/N
2180 NEXT J
2190 RETURN

```

Az alprogram minden elemre megkeresi az első olyan osztópontot, amelynél az elem kisebb vagy egyenlő, és az elemet abba az osztályba sorolja, amelyet ez az osztópont felülről határol.

Ha ezt az osztópontot megtalálta, a J1 változó értékét 1-re állítja, és kiugrik a belső ciklusból.

Ha a ciklus végén a J1 változó értéke 0 maradt, az adatot nem sikerült besorolni, mert nincs nála nagyobb vagy vele egyenlő osztópont. Ebben az esetben az adat a K+1-dik osztályba tartozik (2140-es programsor).

Az első osztópontnál kisebb vagy azzal egyenlő elemek a nulladik osztályba kerültek.

2.3 KERESÉS

Ha egy vektorban tárolt (numerikus vagy karakteres) adatok rendezettségéről semmilyen előzetes információ nincs, akkor egy bizonyos adat megkeresésére egyetlen lehetőség kínálkozik: egyenként végig kell vizsgálni az elemeket, és össze kell hasonlítani a keresett elemmel.

Ha biztosak vagyunk abban, hogy a keresett adattal a vektornak legfeljebb egy eleme egyezik meg, akkor a keresés véget ér, amint ezt az elemet megtaláltuk.

A következő alprogram programozástechnikailag a keresési feladatok legegyszerűbb megoldása. Az A\$ vektorban tárolt nevek között megkeresi a K\$ tartalmával egyezőt, és ennek indexét visszaadja a K változóban.

```

1000 REM **** 55.PROGRAM ****
1010 REM A$(N)-A NEVEKET TARTALMAZO VEKTOR
1020 REM K$----A KERESETT NEV
1030 REM K-----A MEGTALALT NEV INDEXE
1040 REM HA ERTEKE NULLA A KERESÉS EREDMENYTELEN
1050 :
1060 K=0
1070 FOR I=1 TO N
1080 IF K$=A$(I) THEN K=I:RETURN
1090 NEXT I
1100 RETURN

```

Ha az adathalmaz több azonos elemet is tartalmaz, akkor a keresés nem fejeződik be az első elem megtalálása után. A program szerkezetét ebben az esetben az határozza meg, hogy mi volt a célja a keresésnek. Tegyük fel pl., hogy egy személyi nyilvántartásban egy-egy személy adatait a következő vektorokban tároljuk:

N\$(I) — az *i*-dik személy neve;
E(I) — az *i*-dik személy születési éve;
B\$(I) — az *i*-dik személy beosztása;
F(I) — az *i*-dik személy fizetése.

Feladatunk egy olyan alprogram készítése, amely kiírja kívánság szerint képernyőre vagy nyomtatóra egy adott évben született személyek adatait.

Az alprogram megírásakor feltételeztük, hogy a felsorolt vektorok már a tárban vannak.

```

1000 REM **** 56.PROGRAM ****
1010 REM KERESÉS,LISTÁZÁS
1020 REM A$--ELVALASZTO SOR
1030 REM E---A KERESETT SZULETESI EV
1040 :
1050 :
1060 :
1070 A$=" "
1080 PRINT"KIRAS KEPERNYORE--->K"
1090 PRINT"KIIRAS NYOMIATORA--->N"
1100 GETG$:IFG$="OR(G$<)"K"ANDG$<"N">THEN1100
1110 IF G$="K" THEN P=3:GOTO1130
1120 P=4
1130 OPEN1.P
1140 FOR I=1 TO N
1150 IF E<E(I) THEN 1240
1160 PRINT#1,"";N$(I)
1170 PRINT#1,A$
1180 PRINT#1,"SZULETESI EV ";E(I)
1190 PRINT#1,"BEOSZTAS ";B$(I)
1200 PRINT#1,"FIZETES ";F(I)
1210 PRINT#1,A$
1220 PRINT"BARMELY BILLENITYURE TOVABB!"
1230 GETG$:IFG$=" "THEN 1230
1240 NEXT I
1250 CLOSE1
1260 RETURN

```


A feladat megoldását nagyon leegyszerűsítette az a tény, hogy a kikeresett adatokkal semmilyen feldolgozást nem kellett végezni azon kívül, hogy az adott indexhez tartozó adatokat kiírtuk a képernyőre vagy a nyomtatóra. Elképzelhető azonban, hogy a feldolgozás jóval bonyolultabb egyetlen (vagy néhány) PRINT utasításnál, és esetleg az már nem is az éppen futó program feladata. Ilyenkor a kereső alprogram tárolhatja (pl. lemezen) a kikeresett adatok indexeit:

```
1000 REM **** 57.PROGRAM ****
1010 REM KERESÉS-TÁROLÁS FELDOLGOZÁSHOZ
1020 REM E-----A KERESETT SZÜLETESI ÉV
1030 REM E(N)---A SZÜLETESI ÉVEKET
1040 REM          TARTALMAZÓ VEKTOR
1050 REM K(N/2)-A KERESETT ÉVBEN SZÜLETETT SZEMÉLYEK SORSZÁMAI
1060 :
1070 :
1080 :
1090 J=0
1100 FOR I=1 TO N
1110 IF E=E(I) THEN J=J+1:K(J)=I
1120 NEXT I
1130 RETURN
```

Az alprogram rendre végigvizsgálja a nyilvántartást és az adott évben született személyek adatainak indexét tárolja a K vektorban. A J változó a rutin lefutása után az adott évben született személyek létszámát tartalmazza. Az 1050-es REM sorban megjelölt K(N/2) arra utal, hogy a K tömböt N/2 elemszámmra kell deklarálni, azaz feltételeztük, hogy a nyilvántartásban szereplő személyek közül legfeljebb a teljes létszám fele született az adott évben. A K tömb felső indexhatárát mindig az adathalmaz szerkezetétől függően kell megállapítani.

Az alprogram lefutása után a kikeresett adatokkal a K tömbben tárolt indexek alapján bármilyen feldolgozást elvégezhetünk. A következő alprogramban pl. meghatározzuk az adott évben született személyek átlagfizetését.

```
2000 REM **** 58.PROGRAM ****
2010 REM A KIKERESETT ADATOK FELDOLGOZÁSA
2020 REM J-----AZ ADOTT ÉVBEN SZÜLETETT
2030 REM          SZEMÉLYEK SZÁMA
2040 REM K(J)-A KERESETT SZEMÉLYEK SORSZÁMAI
2050 REM S-----A KERESETT SZEMÉLYEK FIZETÉSENEK ÁTLAGA
2060 :
2070 :
2080 :
2090 :
2100 IF J=0 THEN RETURN
2110 FOR I=1 TO J
2120 S=S+F(K(I))
2130 NEXT I
2140 S=S/J
2150 RETURN
```

A 2100-as programsorban külön kezeljük azt az esetet, amikor az előző alprogram nem talál olyan személyt, aki a keresett évben született volna. Egyébként ui. ebben

az esetben a 2140-es programsorban J=0 miatt a program futása „nullával való osztás” hibaüzenettel megszakadna.

A bemutatott keresési eljárások programozása ugyan nagyon egyszerű, a megoldás azonban mégsem mindig célravezető. Nagy adathalmazok esetén az elemek egyenkénti vizsgálata rendkívül sok időt igényel, és ha a számítógép csak ilyen módszerrel tudna dolgozni, egy ügyes titkárnő talán gyorsabban szolgáltatná az adatokat a főnökének, mint a számítógép.

A programok kidolgozása előtt abból a feltevésből indultunk ki, hogy az adathalmaz rendezettségi viszonyairól semmilyen információnk nincs.

Abban az esetben, ha tudjuk, hogy az adathalmaz valamilyen szempont (rendezési kulcs) szerint rendezett, akkor sokkal hatékonyabb keresési eljárások programozására is van lehetőségünk. Az előbbi megoldásban ahhoz, hogy egy n elemet tartalmazó halmazban egy adatot megkeressünk, a legrosszabb esetben (amikor a keresett adat az utolsó helyen áll) n összehasonlítási műveletet kellett elvégeznünk. Ez a szám jelentősen lecsökkenthető, ha tudjuk, hogy az adathalmaz a keresési szempont szerint rendezett.

2.3.1 Bináris keresés

Tegyük fel, hogy az a szám eleme az

$$a_1, a_2, \dots, a_i, \dots, a_n$$

növekvő sorba rendezett számhalmaznak. Határozzuk meg az a szám indexét az ún. bináris keresési algoritmussal, amely a szükséges összehasonlítások számát tekintve lényegesen hatékonyabb, és ennél fogva lényegesen gyorsabb eljárás az előzőnél.

Az egyszerűség kedvéért tegyük fel, hogy n felírható 2 hatványaként, azaz van olyan k természetes szám, amelyre

$$n=2^k$$

Ez a feltevés megkönnyíti a szükséges összehasonlítások számának becslését anélkül, hogy a becslés eredményét meghamisítaná. (Ha ui. n nem írható fel 2 hatványaként, akkor gondolatban kiegészíthetjük az adathalmaz fiktív elemekkel, és a valódi elemszámot helyettesíthetjük 2-nek az n -hez legközelebb eső, nála nagyobb hatványával. Az így kapott becslés csak nagyobb lehet, mint a tényleges összehasonlítások száma, hibát tehát biztosan nem követünk el.)

Első lépésként felezzük meg az adathalmazt, és az $\frac{n}{2}$ -edik elemet hasonlítsuk össze a keresett számmal. Minthogy az adathalmaz elemei növekvő sorban követik egymást, az összehasonlítás eredményeként az alábbi következtetések valamelyikére juthatunk:

- a) a két elem egyenlő;
- b) az a szám a halmaz felső felében van;
- c) az a szám a halmaz alsó felében van.

Az első esetben a keresett számot megtaláltuk. A második és harmadik esetben a keresést folytatni kell az adathalmaz alsó, ill. felső felében. A megfelelő rész tehát ismét megfelezzük és a középső elemet ismét összehasonlítjuk a keresettel. Mivel fel-

tettük, hogy $n=2^k$, az eljárás legfeljebb k lépésben véget ér, hiszen akkor a felezendő adathalmaz egyetlen elemből áll.

A szükséges összehasonlítások számát így a

$$k = \log_2 n$$

értékkel becsülhetjük, ami jóval kevesebb, mint az előző eljárásban kapott n . Kis elemszám mellett természetesen — tekintve a tárműveletek sebességét — a két különböző eljárással megoldott keresés végrehajtási sebessége között alig észleltünk különbséget. Azonban ha pl. $n=1024$ akkor egyáltalán nem mindegy, hogy a legrosszabb esetben $n=1024$ vagy $k=10$ összehasonlítást kell a gépnek elvégezni.

A következő alprogram meghatározza az X szám indexét az A vektorban. A programban nem volt szükség az $n=2^k$ feltevésre.

```
1000 REM **** 59.PROGRAM ****
1010 REM BINARIS KERESÉS
1020 REM A(N)---A SZÁMOKAT TARTALMAZÓ
1030 REM RENDEZETT VEKTOR
1040 REM X-----A KERESÉTT SZÁM
1050 REM K-----A KERESÉTT SZÁM INDEXE
1060 REM N1,N2---AZ AKTUALIS RESZHALMAZ ALSÓ ES FELSO
1070 REM INDEXHATÁRA
1080 :
1090 :
1100 :
1110 K=0
1120 N1=1:N2=N
1130 L=INT((N1+N2)/2)
1140 IF X=A(L) THEN K=L:RETURN
1150 IF X<A(L) THEN N2=L:GOTO1170
1160 N1=L
1170 IF N2-N1>1 THEN 1130
1180 IF A(N1)=X THEN K=N1:RETURN
1190 IF A(N2)=X THEN K=N2:RETURN
1200 RETURN
```

Az alprogramban külön kezeltük azt az esetet, amikor a felezésre váró halmaz már csak kételemű (1180-as és 1190-es sorok). Ha ezt a két programsort kihagytuk volna, akkor bizonyos esetekben a továbbfelezés végtelen ciklushoz vezethetne.

Kövessük végig az alprogram működését úgy, hogy az 1170-es sorba egy GOTO 1130 utasítást írunk, tehát a kételemű halmaz vizsgálatát kihagyjuk. Legyen az adathalmaz $A(1)=1$, $A(2)=2$, $A(3)=3$ és $A(4)=4$, a keresett eleme pedig: $X=4$. A változók aktuális értékét minden lépésben írjuk egy táblázatba:

Lépés	N1	N2	L	A(L)	Elágazás
1.	1	4	2	2	$4 \neq 2$ és $4 > 2$ 1160, 1130
2.	2	4	3	3	$4 \neq 3$ és $4 > 3$ 1160, 1130
3.	3	4	3	3	$4 \neq 3$ és $4 > 3$ 1160, 1130
4.	3	4	3	3	
	.	.	.	.	
	.	.	.	.	
	.	.	.	.	

A táblázatból látható, hogy $N1$ és $N2$ értéke a harmadik lépéstől kezdve nem változik, így az összehasonlítandó elemek sem, tehát az alprogram futása „soha” nem érne véget.

2.4 RENDEZŐELJÁRÁSOK

Az előzőekből láttuk, hogy az adathalmaz rendezettség nagyságrenddel csökkenti az adatok visszakereséséhez szükséges műveletek számát, és egyben a végrehajtási időt.

A tárgyalt keresési algoritmus kapcsán feltétlenül ki kell emelni az adathalmaz rendezettségéből eredő további előnyt: az egy elem visszakereséséhez szükséges összehasonlítások becsült számát a rendezettség n -ről $\log_2 n$ -re csökkentette. Ez már önmagában is jelentős hatékonyságnövekedés, de ez nem minden. Rendezetlen adathalmaz esetén, ha több adatot keresünk egymás után, akkor az adathalmazt minden új elem megkereséséhez elejétől végig újra át kell vizsgálni. Ezzel szemben, ha az adatok rendezettek, akkor az első keresett elem indexe alapján kiderül, hogy hol kell keresni a következőt.

Mindezek alapján érthető, hogy a rendezés az egyik legfontosabb programozási feladat. Az a számítástechnikában járatanok számára is világos, hogy egy olyan telefonkönyvben, amelyben a nevek nem ábécé rendben követik egymást, egy adott név alapján a telefonszám visszakeresése szinte lehetetlen vállalkozás. Azt azonban már csak a szakma beavatottjai tudják, hogy a rendezés elemi lépéseinek meghatározása, azaz egy rendezési algoritmus kidolgozása messze nem olyan egyszerű feladat, mint ahogyan azt első pillantásra feltételeznénk. Számtalan rendezési algoritmus létezik. Aligha akad olyan programozó szakember, aki ne próbált volna meg az általa ismert algoritmusoknál jobbat találni. Hozzá kell természetesen tennünk, hogy a korszerű számítógépeken a rendezőprogramok többnyire készen, rendszer-szolgáltatásként a programozó rendelkezésére állnak.

Azt, hogy egy adott feladat megoldására milyen rendezőalgoritmust használjunk, mindig a feladat jellege határozza meg. A rendezési algoritmusok többféleképpen osztályozhatóak.

A rendezési szempont kiválasztása alapján a rendezéseket alapvetően két csoportba, numerikus és alfanumerikus rendezésekre osztjuk.

Aszerint, hogy az adathalmaz elfér-e egyszerre a központi tárban, vagy kénytelenek vagyunk a rendezéshez háttértárolót (pl. lemezegységet) igénybe venni, megkülönböztetnek ún. *belső* (tárban rendező) és *külső rendező eljárásokat*. A tárkapacitás szempontjából jelentős tulajdonsága az egyes algoritmusoknak, hogy egy vektort a saját helyükön vagy segédvektorok felvételével rendeznek.

Az algoritmusok szerkezete szerint három alapvető rendezési eljárástípust különböztetünk meg:

- kiválasztásos rendezések;
- beszúrásos rendezések;
- az elemek felcserélésén alapuló rendezések.

Az, hogy az elemeket növekvő vagy csökkenő sorba rendezzük, ill. hogy numerikus vagy alfanumerikus adathalmazokkal dolgozunk, az algoritmus lényegét nem érinti. Ezért a továbbiakban, ahol külön megjegyzést nem teszünk, az

$$a_1, a_2, \dots, a_n$$

számsorozat növekvő sorba rendezését tekintjük alapfeladatnak.

2.4.1 Kiválasztásos rendezés

Logikai felépítését tekintve ez a rendezési típus a legegyszerűbb. Alapgondolata a következő: válasszuk ki a vektor legkisebb elemét, és azt cseréljük fel az adathalmaz első elemével (hacsak nem az első elem a legkisebb).

Ezt követően a maradék $n-1$ elemből válasszuk ki a legkisebb elemet és ezt cseréljük fel a vektor második elemével. Az eljárást folytatjuk addig, amíg az utolsó elem is a helyére nem került. A program elkészítéséhez felhasználhatjuk a korábban megírt 50. alprogramot, amely meghatározza egy vektor minimális elemét. A programlépések megértését, lényegesen megkönnyíti, ha egy konkrét számhalmaz átalakításának lépéseit végigkövetjük.

A rendezendő halmaz: 5, 7, 1, 4, 9, 2

	1.	2.	3.	4.	5.	6.	Minimális elem
1. lépés	5	7	1	4	9	2	1
2. lépés	1	7	5	4	9	2	2
3. lépés	1	2	5	4	9	7	4
4. lépés	1	2	4	5	9	7	5
5. lépés	1	2	4	5	9	7	7
6. lépés	1	2	4	5	7	9	

A függőleges vonal a rendezett részhalmazt választja el.

```

1000 REM **** 50. PROGRAM ****
1010 REM KIVÁLASZTÁSOS RENDEZÉS
1020 REM A(N)---A RENDEZENDO VEKTOR
1030 :
1040 :
1050 :
1060 FOR K=1 TO N-1
1070 GOSUB 2000: IF K=J THEN 1090
1080 X=A(J):A(J)=A(K):A(K)=X:REM CSERE
1090 NEXT K
1100 RETURN
1110 :
1120 :
1130 :
2000 REM A VEKTOR MINIMALIS ELEMENEK MEGHATÁROZÁSA
2010 REM A(N)---A VEKTOR
2020 REM MI-----A MINIMALIS ELEM
2030 :
2040 :
2050 MI=A(K):J=K
2060 FOR I=K+1 TO N
2070 IF A(I)<MI THEN MI=A(I):J=I
2080 NEXT I
2090 RETURN

```

A kiválasztásos rendezési algoritmus az $A(N)$ tömböt a saját helyén rendezi, se-gédtömböt nem igényel. Becsüljük meg a rendezéshez szükséges összehasonlítások számát! Az első elemet a minimális elem kiválasztásához $n-1$ elemmel kell összehasonlítani, a következőt $n-2$ elemmel, és így tovább, míg végül az $n-1$ -edik elemet 1 elemmel. Ezek összege:

$$n-1+n-2+n-3+\dots+1=\frac{n(n-1)}{2}=\binom{n}{2}.$$

A szükséges összehasonlítások száma az elemszám négyzetével arányosan nő, ami arra utal, hogy ezt a módszert nagy elemszám esetén nem célszerű alkalmazni.

2.4.2 Beszúrásos rendezés

Tegyük fel, hogy be kell olvasnunk billentyűzetről n személy nevét. A nevek a fel-dolgozás során ábécé rendben kívánunk hivatkozni. A gépkezelőt nem akarjuk arra kényszeríteni, hogy begépelés előtt kézzel névsorba rendezzen, de az adatbevitelt jó lenne úgy megszervezni, hogy a teljes névsor begépelése után az adatok már ábécé rendben legyenek. A feladat tipikus esete a beszúrásos rendezésnek. Az első nevet beolvassuk, és az így kapott egyelemű halmazt rendezettnek tekintjük. A soron kö-vetkező nevet be kell illesztenünk a névsorba úgy, hogy az az ábécé sorrend szerint a helyére kerüljön. A beszúrásos rendezés alaplépése: egy tetszőleges elemet be kell illeszteni egy már rendezett részhalmazba. Miután az összehasonlítások sorozatával megállapítottuk a soron következő elem helyét, az ezt követő elemeket rendre egy hellyel „feljebb” kell csúsztatnunk az adathalmazban. Szemléltessük a beszúrást egy betűkből álló halmazon!

Sorszám	1.	2.	3.	4.	5.	6.	7.
	A	G	H	J	L		
A beszúrásos betű: B A beszúrás eredménye:							
Sorszám	1.	2.	3.	4.	5.	6.	7.
	A	B	G	H	J	L	

Az egy elem beszúrásához szükséges összehasonlítások számát most nem tudjuk pontosan meghatározni, értéke ui. a nevek begépelésének véletlen sorrendjétől függ. A karakteres összehasonlítás a Commodore 64-esen a numerikus összehasonlítás-hoz hasonlóan végezhető el, a programírás során erre nem kell külön figyelmet for-dítani.

Célszerű azonban a soron következő elem beillesztését külön alprogramban elvé-gezni.

Az alprogramokban használt fontosabb változók:

- I — az éppen beolvasott név sorszáma;
- XS — a beolvasott név vagy lezáró (*) karakter;
- NS — a neveket rendezetten tartalmazó vektor;
- J — a beszúrás helye.


```

1000 REM **** 61.PROGRAM ****
1010 REM BESZURASOS RENDEZES ADATBEVITEL KOZBEN
1020 REM N$(N)---A NEVEK VEKTORA
1030 REM J-----A BEOLVASOTT NEV HELYE A VEKTORBAN
1040 :
1050 :
1060 :
1070 I=1
1080 PRINT "AZ ADATBEVITEL VEGET GAL JELEZZE!"
1090 PRINT I".NEV";
1100 INPUT X$: IF X$="" THEN RETURN
1110 IF I=1 THEN N$(I)=X$:GOTO1170
1120 FOR J=1 TO I-1
1130 IF N$(J)>X$ THEN 1150
1140 NEXT J
1150 REM J----A BESZURAS HELYE
1160 GOSUB2000
1170 I=I+1:GOTO1090
1180 :
1190 :
1200 :
2000 REM **** BESZURAS ****
2010 IF J=I THEN 2050:REM AZ ELEM AZ UTOLSO HELYRE KERUL
2020 FOR K=1 TO J+1 STEP-1
2030 N$(K)=N$(K-1)
2040 NEXT K
2050 N$(J)=X$
2060 RETURN

```

Első látásra talán nem világos, hogy miért kellett az éppen beolvasott elem beszúrása során a 2000-es alprogramban az elemek hátracsúztatását hátulról kezdeni.

Nézzük meg a feladatot egy konkrét példán keresztül! Az egyszerűség kedvéért jelöljük a neveket kezdőbetűkkel! Az eddig tárolt nevek:

1.	2.	3.	4.	5.
A	D	H	L	M

A beszúrandó betű: E, sorszáma (az I változó) 6. Az 1120-es sortól az 1140-es sorig terjedő vizsgálatban a J változó értéke 3 lesz, hiszen ez az első olyan index, amelyre teljesül az

$$N$(J) > X$$

feltétel, ahol N\$(J) a már rendezett névsor soron következő eleme, X pedig a beszúrandó név.

A neveket a harmadik elemtől kezdve eggyel jobbra kell csúztatnunk és a harmadik helyre az E nevet kell beírunk. Az aktuális részhalmaz:

	3.	4.	5.	6.
beszúrás előtt	H	L	M	
beszúrás után	E	H	L	M

A következő ciklusban a J+1-edik elemtől kezdve minden elem helyére betesszük az eredeti sorrendben előtte álló-, majd a J-edik helyre az éppen beolvasott elemet.

Valóban a várt eredményt kapjuk? Nem egészen:

	3.	4.	5.	6.
beszúrás előtt	H	L	M	
beszúrás után	E	H	H	H

Az $N(K) = N(K-1)$ értékadással minden lépésben elveszítenénk a soron következő elem tartalmát.

Mint hogy a szükséges összehasonlítások pontos értékét nem tudjuk meghatározni, adjunk rá felső becslést.

Az i -dik lépésben legrosszabb esetben (ha a beszúrandó elem nagyságrendben az eddigiek közül a legnagyobb) $i-1$ összehasonlítást kell elvégeznünk. Az összehasonlítások maximális száma tehát a beszúrásos rendezés esetében is az elemek számának négyzetével arányos. Jelentősen csökkenthető ez az érték, ha az elem helyének meghatározását nem lineáris (egymást követő elemekkel való) összehasonlítással, hanem bináris kereséssel határozzuk meg. A bináris keresésnél láttuk, hogy egy i elemű halmazban az i -edik elem helyét maximum

$$\log_2 i$$

számú összehasonlítással találhatjuk meg. Ha ezt az értéket összegezzük $i=2$ -től n -ig, akkor az összehasonlítások számára a következő felső becslést nyerjük:

$$\sum_{i=2}^n \log_2 i = \log_2 2 + \log_2 3 + \log_2 4 + \dots + \log_2 n \approx n \log_2 n.$$

Ha az elem helyét bináris kereséssel határozzuk meg, akkor n elem beszúrásos rendezéséhez durva becsléssel legfeljebb

$$n \log_2 n$$

összehasonlítást kell végeznünk. A felső becslés alapján a bináris kereséssel javított beszúrásos rendezés lényegesen hatékonyabbnak tűnik, mint a kiválasztó rendezés. Nem szabad azonban elfeledkezni arról, hogy a javítás csak az összehasonlítások számára vonatkozik, az adatok mozgatására (csúztatásra) nem. A beszűrő rendezés a javítás ellenére sem lényegesen jobb eljárás a kiválasztó rendezésnél.

Ha a rendezendő tömb a tárban van (nem beolvasás közben rendezünk), akkor a rendezéshez egy — az eredeti tömbbel azonos méretű — segéd tömböt is igénybe kell vennünk.

2.4.3 A buborékredezés

A buborékredezés az elemek cseréjén alapuló rendezőalgoritmusok közül a legegyszerűbb. Elnevezése tükrözi az eljárás lényegét: ha az elemeket csökkenő sorba rendezzük, egy lépéssorozatban a legkisebb elem — a víz felszínére felszálló buborékhoz hasonlóan — a vektor legmagasabb indexű helyére kerül. Az első lépéssorozatban rendre összehasonlítjuk az első elemet a másodikkal, a másodikat a harmadikkal, és így tovább, mindaddig, amíg az utolsó előtti elemhez nem érünk. Összehasonlítás után, ha szükséges, a két szomszédos elemet felcseréljük. Ha elértük a vektor legkisebb elemét, a következő összehasonlítások során ez az elem minden esetben egy hellyel jobbra mozdul, mindaddig, amíg az utolsó helyre nem érkezik.

Szemléltetésül ábrázoljunk egy ilyen lépéssorozatot!

Az eredeti vektor: 7 1 3 2 5 8

1. lépés 7 1 3 2 5 8 cseré nem történt
2. lépés 7 3 1 2 5 8 a 2. és a 3. elem helyet cserél
3. lépés 7 3 2 1 5 8 a 3. és a 4. elem helyet cserél
4. lépés 7 3 2 5 1 8 a 4. és az 5. elem helyet cserél
5. lépés 7 3 2 5 8 1 az 5. és a 6. elem helyet cserél

A vonal az összehasonlításra utal.

Az elvégzett lépéssorozat után egy biztos információnk van az adathalmaz rendezettségéről: a legkisebb elem biztosan a helyére, nevezetesen a sor végére, az n -edik helyre került. A következő lépéssorozatban elegendő tehát az elemeket az $n-1$ -edik elemig páronként összehasonlítani, ill. szükség esetén felcserélni.

A buborékredezés alprogramja:

```
1000 REM **** 62.PROGRAM ****
1010 REM EGYSZERU BUBOREKRENDEZES
1020 REM A(N)---A RENDEZENDO VEKTOR
1030 :
1040 :
1050 :
1060 FOR K=N-1 TO 1 STEP-1
1070 FOR I=1 TO K
1080 IF A(I)>A(I+1) THEN 1100
1090 A=A(I):A(I)=A(I+1):A(I+1)=A:REM CSERE
1100 NEXT I
1110 NEXT K
1120 RETURN
```

A szükséges összehasonlítások számát ismét könnyen meghatározhatjuk. Az első lépésben $n-1$, a következő lépésben $n-2$ és így tovább, míg végül az utolsó lépésben egyetlen összehasonlításra van szükség, így az összehasonlítások maximális száma:

$$n-1+n-2+\dots+2+1=\frac{n(n-1)}{2}=\binom{n}{2}$$

Ezt az értéket jelentősen csökkenthetjük, és így a buborékredezés hatékonyságát jelentősen megnövelhetjük, ha figyeljük, hogy egy adott lépéssorozatban volt-e csere vagy sem. Ha ui. nem történt csere, akkor a vektor elemei már tökéletes rendben követik egymást, az eljárást befejezhetjük.

A cserék számának figyeléséhez az alprogramot a következők szerint kell kiegészíteni:

```
1000 REM **** 62/A.PROGRAM ****
1010 REM EGYSZERU BUBOREKRENDEZES
1020 REM A(N)---A RENDEZENDO VEKTOR
1030 :
1040 :
1050 :
1060 FOR K=N-1 TO 1 STEP-1
1065 C=0
1070 FOR I=1 TO K
1080 IF A(I)>A(I+1) THEN 1100
1090 A=A(I):A(I)=A(I+1):A(I+1)=A:C=C+1:REM CSERE
1100 NEXT I
1105 IF C=0 THEN RETURN
1110 NEXT K
1120 RETURN
```

Azt természetesen nem lehet előre pontosan meghatározni, hogy a módosítás mennyivel csökkenti az összehasonlítások számát, hiszen az a vektor elemeinek előzetes rendezettségi állapotától függ, ami véletlenszerű. Ha pl. az eredeti adathalmaz pontosan a kívánttal fordított sorrendű, akkor az összehasonlítások száma az előbbi módosítással sem csökkenthető. Szemben azonban a korábban ismertetett algoritmusokkal, az utóbbi „azonnal észreveszi”, ha már tökéletesen rendezett adathalmazzal dolgozik. Sem a beszűrő rendezésnek, sem a kiválasztásos rendezésnek nincs meg ez a tulajdonsága.

Az utóbbiak az egy elem elhelyezéséhez szükséges lépéssorozatot az adathalmaz előzetes rendezettségi viszonyaitól függetlenül $n-1$ -szer végrehajtják.

2.4.4 A gyorsrendező

A rendezőalgoritmusok hatékonyságának becsléséhez nem elegendő az összehasonlítások számát megvizsgálni, hiszen pl. a beszűrő rendezés során jelentős mennyiségű műveletidőt vesz igénybe a tömb részleges vagy teljes csúsztatása, az elemek cseréjén alapuló eljárásokban pedig a cserélgetés. A gyorsrendező algoritmus kifejlesztése éppen azon a megfigyelésen alapszik, hogy a cserék számát jelentősen lehet csökkenteni, ha nem az egymás melletti, hanem az egymástól távol eső elemeket cserélgetjük egymással. Ha az elemek sorrendje a kívánttal éppen ellentétes, akkor egyáltalán nem mindegy, hogy rendre az egymástól legtávolabb eső elemeket cseréljük fel, vagy a szomszédosakat. Az

1, 2, 3, ..., $n-1$, n

halmaz fordított sorrendjéhez mindössze $\frac{n}{2}$ cserére és összehasonlításra van szük-

ség, ha a halmaz két széléről befelé haladunk. Ugyanezt az adathalmazt a javított buborékrendezés a szomszédos elemek összehasonlításával és cseréjével

$$\frac{n(n-1)}{2}$$

művelet elvégzésével rendezné.

A következőkben ismertetett algoritmus hatékonysága az eddigiek hatékonyságához képest annyira szembeötlő, hogy megalkotója (C. A. R. Hoare) méltán nevezte gyorsrendező eljárásnak. Az algoritmus ismertetése előtt meg kell jegyeznünk, hogy a hatékonyság ára az algoritmus bonyolultsága (ez természetesen nem általános igazság). Programozástechnikailag ez a módszer sokkal összetettebb, mint az előzőek voltak. Alapfeladatunk legyen ismét n szám növekvő sorba rendezése.

A gyorsrendező algoritmus lényege a következő: válasszuk ki a vektor középső (páros n esetén az $\frac{n}{2}$ -edik, páratlan n esetén az $(\frac{n}{2}+1)$ -edik elemét, és osszuk a vektort két részhalmazra úgy, hogy az egyikbe az adott elemnél kisebb, esetleg egyenlő, a másikba az adott elemnél nagyobb vagy esetleg egyenlő elemek kerüljenek! A kapott két részhalmaz mindegyikével ismételjük meg ezt a lépést!

Minthogy minden lépésben két részhalmazt kapunk, és egyszerre csak egy részhalmaz átrendezésével tudunk foglalkozni, az egyiket mindig „tegyük félre”, és csak akkor térjünk vissza rá, amikor az aktuális részhalmaz már csak egy elemből áll. A félretett részhalmazok száma gyarapodni fog mindaddig, amíg „rá nem érünk” újra foglalkozni velük. Mivel a kisebb részhalmazokkal gyorsabban lehet végezni, állapodjunk meg abban, hogy a keletkezett részek közül mindig a nagyobbát „teszszük félre”. Mielőtt a folyamatot megpróbálnánk egy példával szemléltetni, meg kell fontolnunk, hogy a csoportosítást milyen módszerrel végezzük el.

Emeljük ki a középső elemet! Induljunk el a vektor legelső elemétől, és az elemeket rendre összehasonlítva a középső elemmel haladjunk mindaddig, amíg nála nagyobbbat vagy egyenlőt nem találunk! Az ilyen elem ui. rossz helyen van, át kell kerülnie a felső részhalmazba. Hogyan?

Jegyezzük meg a „rendbontó” elem helyét, és most lépegezzünk egyesével az utolsó elemtől visszafelé mindaddig, amíg a középső elemnél kisebbet nem találunk. Ez is rossz helyen van. A kiválasztott két elemet most cseréljük fel, és induljunk el ismét lentről felfelé, majd fentről lefelé, onnan, ahol az előbb megálltunk. Az alsó és a felső pozíciót tároljuk egy-egy változóban, és értékeket egyesével növeljük. Az eljárást mindaddig folytatjuk, amíg a két mutató egyenlő nem lesz, ill. a még az alsó túl nem lép a felsőn vagy megfordítva. Az elmondottakat szemléltessük egy példával úgy, hogy a pillanatnyi pozíciót egy $\uparrow$ és egy $\downarrow$ nyíllal ábrázoljuk, a kiszemelt elemet pedig bekeretezzük. A félretett részhalmazok alsó és felső határának indexét egy-egy veremben tároljuk.

Megjegyzés: a verem (stack) alapvető számítástechnikai fogalom. Olyan átmeneti tárolót jelöl, amelyből az elemeket a behelyezésükkel ellentétes sorrendben vesszük elő. A számítástechnikai szakirodalomban gyakran találkozunk a *last in, first out* („ami utoljára be, az először ki”) angol kifejezéssel, ami a verem fogalmának tömör meghatározása.

Lépések: a lépéseket 1-től számozzuk, valahányszor egy új átrendezéshez kezdünk.

1. 6 2 7 3 1 9 4 $\downarrow$
 $\uparrow$

Cserélendő elemek: 6, 1

2. 1 2 7 3 1 9 4 $\downarrow$ 6
 $\uparrow$

3. 1 2 7 3 1 9 4 $\downarrow$ 6
 $\uparrow$

4. 1 2 7 3 1 9 4 $\downarrow$ 6
 $\uparrow$

5. 1 2 7 3 $\downarrow$ 1 9 4 6
 $\uparrow$

Cserélendő elemek: 7, 1

1 2 1 3 7 9 4 6 4 8

A lépéssorozat végére a két nyíl összeért, a halmaz tehát két részre oszlott. A bekeretezett részhalmazt félretesszük, és megismétljük az eljárást az alsó részhalmazzal.

1. 1 2 $\downarrow$ 1 3 7 9 4 6 A félretett részhalmazok
 $\uparrow$

2. 1 2 1 3 7 9 4 6
 $\uparrow$

Cserélendő elemek: 2, 1

1 $\downarrow$ 2 3 7 9 4 6 1 2
 $\uparrow$ $\uparrow$ 4 8

az alsó és a felső határ indexei

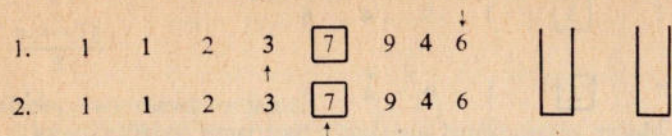
A két nyíl túlhaladt egymáson, a kapott részhalmazok közül a nagyobbát ismét félretesszük.

A „2” egyelemű részhalmaz rendezett, így ezt a munkafázist befejeztük, elővesszük az első félretett részhalmazt:

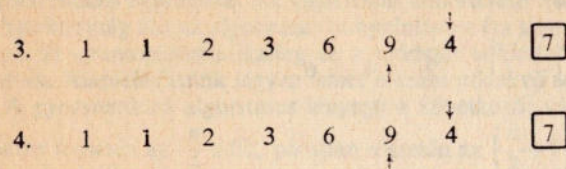
1. 1 $\downarrow$ 2 3 7 9 4 6
 $\uparrow$

2. 1 1 2 3 7 9 4 6 4 8
 $\uparrow$

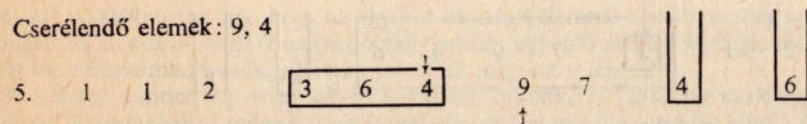
Újabb részhalmaz nem keletkezett, ismét elővesszük a korábban félretett halmazt:



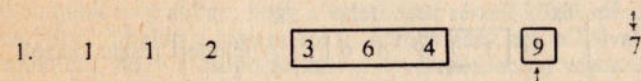
Cserélendő elemek: 7, 6



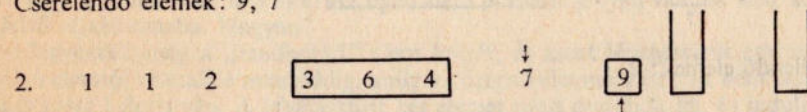
Cserélendő elemek: 9, 4



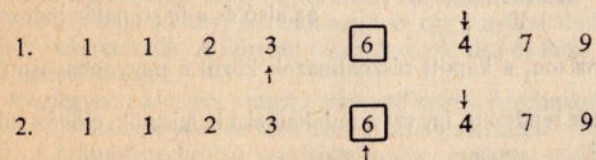
A rendezést ismét a kisebb halmazzal kezdjük.



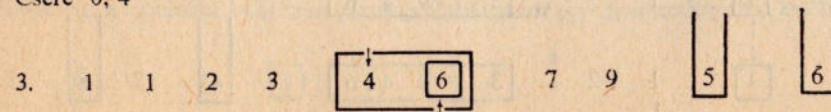
Cserélendő elemek: 9, 7



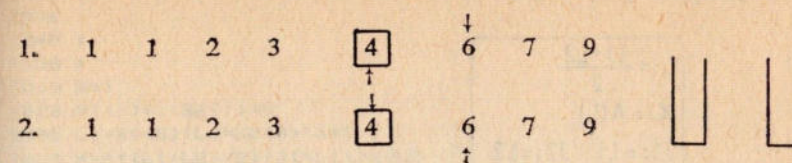
A verem ismét üres, mert az aktuális részhalmazok egyeleműek.



Csere: 6, 4



A keletkezett halmazok egyike ismét egyelemű, így amit félretettünk, azonnal elővesszük:



A verem kiürült, az aktuális halmazok egyeleműek, a rendezéssel készen vagyunk.

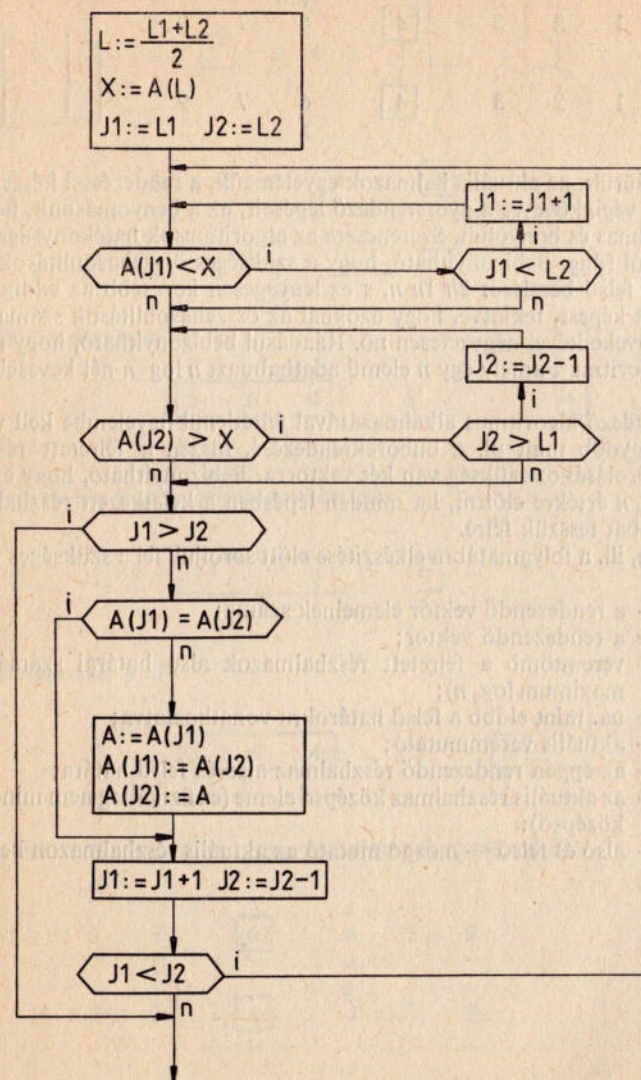
Gondosan végigkövetve a gyorsrendező lépéseit, az a benyomásunk, hogy az eljárás hosszadalmas és bonyolult. Szerencsére az algoritmusok hatékonysága nem a benyomásainktól függ. Bebizonyítható, hogy a szükséges összehasonlítások maximális számának jó felső becslése: $2n \ln n$, s ez lényegesen kevesebb az eddig megismert eljárásokéhoz képest, tekintve, hogy azoknál az összehasonlítások száma az elemek számának növekedtével négyzetesen nő. Ráadásul bebizonyítható, hogy nem is létezik olyan algoritmus, amely egy n elemű adathalmazt $n \log n$ -nél kevesebb lépésben rendezne.

A gyorsrendező algoritmus alkalmazásával feltétlenül figyelembe kell venni, hogy tárigénye nagyobb mint pl. a buborékrendezésé, hiszen a félretett részhalmazok indexeinek tárolásához szükség van két vektorra. Bebizonyítható, hogy ezek méretét elegendő $\log_2 n$ értékre előírni, ha minden lépésben a keletkezett részhalmazok közül a nagyobbát tesszük félre.

A program, ill. a folyamatábra elkészítése előtt soroljuk fel a szükséges változókat:

- N — a rendezendő vektor elemeinek száma;
- A — a rendezendő vektor;
- S1 — veremtömb a félretett részhalmazok alsó határai számára (mérete maximum $\log_2 n$);
- S2 — ua. mint előbb a felső határookra vonatkoztatva;
- S — aktuális veremmutató;
- L1, L2 — az éppen rendezendő részhalmaz alsó és felső határa;
- X — az aktuális részhalmaz középső eleme (ez fizikailag nem minden esetben középső);
- J1, J2 — alsó és felső — mozgó mutató az aktuális részhalmazon belül.

A gyorsrendező algoritmus legkritikusabb része az adott halmaz két részhalmazra bontása. Azok számára, akik vizuálisan jobban tájékozódnak, közöljük a programrészlet folyamatábráját (4. ábra).



4. ábra Részhalmazokra bontás

```

1000 REM **** 63.PROGRAM ****
1010 REM NOVEKVO SORBA RENDEZES GYORSRENDEZOVEL
1020 REM A(N)----A RENDEZENDO VEKTOR
1030 :
1040 :
1050 :
1060 S=1
1070 S1(1)=1:S2(1)=N
1080 L1=S1(S):L2=S2(S):S=S+1
1090 X=A((L1+L2)/2):J1=L1:J2=L2
1100 IF A(J1)<X AND J1<L2 THEN J1=J1+1:GOTO1100
1110 IF A(J2)>X AND J2>L1 THEN J2=J2-1:GOTO1110
1120 IF J1>J2 THEN 1170
1130 IF A(J1)=A(J2) THEN 1150
1140 A=A(J1):A(J1)=A(J2):A(J2)=A
1150 J1=J1+1:J2=J2-1
1160 IF J1<J2 THEN 1100
1170 IF L2-J1>J2-L1 THEN 1200
1180 IF J2>L1 THEN S=S+1:S1(S)=L1:S2(S)=J2
1190 L1=J1:GOTO1220
1200 IF L2>J1 THEN S=S+1:S1(S)=J1:S2(S)=L2
1210 L2=J2
1220 IF L2>L1 THEN 1090
1230 IF S<=0 THEN RETURN
1240 GOTO1080
  
```

Az alprogram leírása

- 1060—1070 Beállítjuk a kezdőértékeket. Az első esetben a vizsgált részhalmaz a teljes vektor. A veremmutató értéke egy, a veremekben tárolt alsó és felső határ 1, ill. N.
- 1080 Egy részhalmaz határait elővesszük a veremből, a veremmutatót eggyel csökkentjük.
- 1090—1220 Az aktuális részhalmaz két részre bontása.
- 1090 Meghatározzuk a középső elemet, a mozgó mutatókat az aktuális részhalmaz alsó és felső határára állítjuk.
- 1100 A halmazon belül az alsó mutatót mindaddig növeljük, amíg a ki-tüntetett elemnél (X) nagyobbat vagy egyenlőt nem találunk. Közben vizsgáljuk, hogy a részhalmaz felső határát nem léptük-e túl (J1 > L2). A halmazon belüli felső mutatót mindaddig csökkentjük, amíg a ki-tüntetett elemnél (X) kisebb vagy egyenlő elemet nem találunk.
- 1110 Ha az alsó mutató nagyobb, mint a felső, akkor a felosztás befeje-ződött, kiugrunk a ciklusból.
- 1120 Ha a két elem egyenlő, akkor csere nélkül tovább lépünk: a felső mu-tatót eggyel csökkentve, az alsót eggyel növelve.
- 1130 A két elemet megcseréljük, és
- 1140 a mutatókat léptetjük.
- 1150 Ha a mutatók még nem találkoztak, vagy nem léptek túl egymáson, akkor folytatjuk a halmaz felosztását.
- 1160 Ha a mutatók még nem találkoztak, vagy nem léptek túl egymáson, akkor folytatjuk a halmaz felosztását.
- 1170—1210 Az egyik részhalmaz határait a veremben, a másik részhalmaz ha-tárait pedig az L1, L2 változóban tároljuk.
- 1170 Ha a felső részhalmaz nagyobb, akkor ugrunk az 1200-as sorra.

- 1180 Az alsó részhalmaz határait helyezzük a verembe, de csak akkor, ha valódi részhalmaz ($J2 > L1$).
- 1190 A rendezendő halmaz felső határa ($L2$) változatlan, alsó határa $J1$.
- 1200—1210 Ua. mint az 1180—1190-es sorokban, de most a felső részhalmaz kerül a verembe, mert ez nagyobb.
- 1220 Ha a rendezendő halmaz valódi részhalmaz, visszatérünk a rendező programrészre.
- 1230 Megnézzük, hogy a verem üres-e, ha igen, készen vagyunk, egyébként (1080-as) visszatérünk a főciklushoz.

Összefoglalásként készítünk egy programot, amellyel összehasonlítjuk az egyes rendezési eljárások végrehajtás idejét. A rendezéshez szükséges időtartamot a gép belső órája segítségével mérjük. Az egyes alprogramok elején a TIS rendszerváltozó értékét nullára állítjuk, majd tartalmát a rendezés után egy tömbben tároljuk. A program megkérdezi (a 130-as programsorban), hogy maximum hány elemre végezze el a rendezést. Első lépésben a tömb elemeinek száma 10. Miután a rendezést minden eljárással végrehajtottuk 10 elemre, az elemek számát megduplázzuk. Az eljárást mindaddig folytatjuk, amíg a kapott elemszám (N) el nem éri a kívánt értéket. Ezt követően kinyomtatjuk az összehasonlító táblázatot.

A rendezendő vektort véletlenszám-generátorral töltjük fel. Miután minden rendezőnek az eredeti számhalmazt kell rendeznie, az eredeti vektort tároljuk egy segédvektorban, és minden rendezés után töltjük vissza!

A rendezőeljárások közül csak a beszűrő rendezés listáját közöljük, hiszen ezt az eljárást eddig csak beolvasás közbeni rendezésre alkalmaztuk. A többi rendezés alprogramjai azonosak a korábban bemutatott alprogramokkal.

A program futásának eredménye:

RENDEZO ELJARASOK OSSZEHASONLITASA

N	10		20		40		80		160		320	
	MM	SS	MM	SS	MM	SS	MM	SS	MM	SS	MM	SS
BESZ.REND.:00	00	00	02	00	08	00	31	02	02	08	02	
KIV.REND.:00	00	00	02	00	06	00	24	01	30	05	48	
BUB.REND.:00	01	00	03	00	12	00	58	03	53	15	57	
JBUB.REND.:00	01	00	04	00	13	01	00	04	05	16	53	
GYORSREND.:00	02	00	04	00	10	00	22	00	51	01	55	

MM: PERC

MM=MASODPERC

A táblázat alapján levonhatunk néhány következtetést:

a) A buborékrendezés a legkevésbé hatékony eljárás. A rendezés végrehajtási idejét jelentősen növeli a többi eljáráshoz képest a cserék száma. A javított buborékrendezés futtatási ideje rosszabb, mint az egyszerű buborékrendezés ideje. Erre könnyen adódik magyarázat: ha az adathalmaz nem volt előrendezett, akkor a javítás a szükséges összehasonlítások és cserék számát nem csökkenti. Ugyanakkor a cserék számának vizsgálata a belső ciklus minden lefutása után egy $IF \dots THEN \dots$ utasítás végrehajtását igényli, amely az egyszerű rendezésben nem szerepel.

b) A kiválasztó rendezés a gyorsrendező után a leghatékonyabb eljárás. Ez is érthető, hiszen jóval kevesebb cserét kell végrehajtania, mint a buborékrendezésnek.

c) A beszűrő rendezés a kiválasztó rendezés mögött maradt. Ennek magyarázata az egy elem beszűréséhez szükséges csúsztatásban rejlik.

Az összehasonlító program listájából csak azon részeket közöljük, amelyek korábban nem fordultak elő.

```

10 REM **** 64.PROGRAM ****
11 :
20 REM RENDEZESEK OSSZEHASONLITASA
30 DIM A(400),B(400),C(400),S1(10),S2(10)
40 REM A(400)---A RENDEZENDO VEKTOR
50 REM B(400)---SEGEDVEKTOR A RENDEZENDO VEKTOR MEGORZESEHESZ
60 REM C(400)---SEGEDVEKTOR A BESZURO RENDEZESHEZ
70 REM S1(10)---SEGEDVEKTOR A GYORSRENDEZOHOS
80 REM S2(10)---SEGEDVEKTOR A GYORSRENDEZOHOS
90 REM N(5)---A RENDEZO ELJARASOK MEGNEVEZESE
100 REM T1-----A FUTTATASOK SZAMA
110 REM A$(I,J)-A VEGREHAJTASI IDOT TARTALMAZO TOMB
120 FOR I=1 TO 5:READ N$(I):NEXT I
130 INPUT "MAXIMALIS ERTEKE(<400)";N1
140 N=10:G=1
150 U$=""
160 FOR I=1 TO N
170 A(I)=INT(RND(0)*1000):B(I)=A(I)
180 NEXT I
190 PRINT "A RENDEZESEK FUTTATASI EREDMENYEI"
210 PRINT "-----"
220 PRINT "AZ ELEMEK SZAMA:";N
230 PRINT "-----"
240 FOR H=1 TO 5
250 ON H GOSUB 1000,2000,3000,4000,5000
260 GOSUB 6000:GOSUB 8000:REM KIIRAS,VISSZAIOLTES
270 NEXT H
280 PRINT "MARMELY BILLETIURE TOVABB!"
290 GET G$:IF G$="" THEN 290
300 N=2*N
310 IF N<=N1 THEN G=G+1:GOTO 160
320 GOSUB 7000:REM NYOMTATAS *****
330 STOP
1000 REM BESZURO RENDEZES *****
1010 :
1020 :
1030 T1$="000000":C(1)=A(1)
1040 FOR I=2 TO N
1050 FOR J=1 TO I-1
1060 IF C(J)>A(I) THEN 1080
1070 NEXT J
1080 GOSUB 1500
1090 NEXT I
1100 A$(H,G)=T1$
1120 RETURN
1130 :
1500 REM BESZURAS *****
1510 IF J=1 THEN 1550
1520 FOR K=1 TO J+1 STEP-1
1530 C(K)=C(K-1)
1540 NEXT K
1550 C(J)=A(I)
1560 RETURN
1570 :

```



```

6000 REM A RENDEZETLEN VEKTOR VISSZATOLTESE***
6010 FOR I=1 TO NIA(I)=B(I):NEXT I
6020 RETURN
7000 REM AZ EREDMENYEK KIIRASA*****
7010 OPEN1,4:FOR U=1TO5
7020 PRINT#1," RENDEZO ELJARASOK OSSZEHASONLITASA"
7030 PRINT#1," "
7040 PRINT#1," "
7050 PRINT#1," N " "N=10:FOR I=1 TO 6
7060 R$=STR$(N):L=LEN(R$):R$=LEFT$(R$,8-L)+R$
7070 PRINT#1,R$:N=2*N
7080 NEXT I:PRINT#1
7090 PRINT#1,SPC(10);" "
7100 FOR I=1TO 6
7110 PRINT#1,"MM";" " "SS";" "
7120 NEXT I:PRINT#1
7130 PRINT#1," "
7140 PRINT#1," "
7150 FOR I=1TO5
7160 PRINT#1,N$(I)+": ";
7170 FOR J=1TO6
7180 P$=MID$(A$(I,J),3,2):S$=MID$(A$(I,J),5,2)
7190 PRINT#1,P$;" "S$;" "
7200 NEXT J:PRINT#1
7210 NEXT I
7220 PRINT#1," "
7230 PRINT#1," "
7240 PRINT#1,"MM:PERC MM=MASODPERC"
7250 GETG$:IFG$=""THEN7250
7260 CLOSE1:RETURN
8000 REM KIIRAS KEPRNYORE*****
8010 PRINT$(H);": ";
8020 PRINTMID$(TI$,3,2);" MIN "MID$(TI$,5,2);" S"
8030 RETURN
9000 DATA BESZ.REND.,KIV. REND.,BUB. REND.
9010 DATA JBUB.REND.,GYORSREND.
9030 RETURN

```

2.5 MÁTRIXOK

2.5.1 Mátrixok beolvasása

A DIM utasítással csak a mátrix nevét és méreteit határozzuk meg, elemei még nem kapnak értéket. A DIM utasítás végrehajtása után a valós és egész szám típusú tömbváltozók értéke nulla, a füzér típusúaké pedig az üres füzér (l. az 1.10 alfejezetben).

Mint minden egyéb változó, a tömbváltó is vagy valamilyen beolvasó, vagy egy értékadó utasítással kaphat értéket. Némi bonyodalmat csak az okoz, hogy hogyan szervezzük meg a programban a mátrix összes elemének beolvasását.

A mátrixok kezeléséhez meg kell ismerkednünk részletesebben a ciklusok egymásba ágyazásával (l. az 1.9 alfejezetben). Kezdjük egy egyszerű példával!

Futtassuk le a következő programot, és nézzük meg, mi lesz a programfutás eredménye!

```

10 REM 65. PROGRAM
20 REM EGYMASBA AGYAZOTT CIKLUSOK
30 :
40 :
50 FOR I=1 TO 2:REM KULSO CIKLUS ELEJE
60 REM KULSO CIKLUS MAGJA
70 FOR J=1 TO 3:REM BELSO CIKLUS KEZDETE
80 PRINT I;J
90 NEXT J : REM BELSO CIKLUS VEGE
100 NEXT I: REM KULSO CIKLUS VEGE

```

A futás eredménye a képernyőn:

```

1 1
1 2
1 3
2 1
2 2
2 3

```

A magyarázat igen egyszerű. A külső ciklussal azt írtuk le, hogy ennek a ciklusnak a magját kétszer kell végrehajtani, mégpedig úgy, hogy I értéke először egy, másodszor kettő lesz. E példában a ciklusmag újra egy ciklus (ez a belső ciklus), amelynek az a feladata, hogy a J értéket egytől kezdve növelje egyesével háromig, és minden cikluslépésben írja ki az I és a J értékét.

A belső ciklus tehát kétszer indul előlről. A 65. programban bemutatott egymásba ágyazott ciklusokkal könnyen megoldható a mátrixok kezelése, ha az I és a J ciklusváltozókat tömbindexként használjuk. Csak a 80-as programsort kell cserélni, attól függően, hogy mit kell csinálnunk a mátrix elemeivel.

Egy 2×3 -as MA nevű mátrix elemeit tehát így olvashatjuk be:

```
10 REM **** 66. PROGRAM ****
20 REM MATRIX BEOLVASAS SORFOLYTONOSAN
30 :
40 :
50 DIM MA(2,3)
60 FOR I=1 TO 2
70 FOR J=1 TO 3
80 PRINT I;"SOR",J;"ELEM KÖVETKEZIK";
90 INPUT MA(I,J)
100 NEXT J
110 NEXT I
```

Az INPUT utasítás előtt célszerű kiírni a beolvasásra kerülő elem indexeit, mert így a program futtatásakor látjuk, hogy melyik elem beolvasása következik.

A program először a MA mátrix első sorát olvassa be

MA(1,1), MA(1,2), MA(1,3) sorrendben

majd a második sorát

MA(2,1), MA(2,2), MA(2,3).

Az ilyen sorrendű beolvasást *sorfolytonos* beolvasásnak nevezzük.

Előfordulhat, hogy valami miatt kényelmesebb először az első oszlop elemeit beolvasni:

MA(1,1), MA(2,1)

majd a második oszlop elemeit:

MA(1,2), MA(2,2)

végül a harmadik oszlopot:

MA(1,3), MA(2,3)

Az ilyen sorrendű beolvasást *oszlopfolytonos* beolvasásnak hívjuk.

Milyen programmal valósítható meg? Legyen most a külső ciklus ciklusváltozója az oszlopindex, és értéke változzon egytől az oszlopok számáig!

Egy oszlopon belül pedig a sorindexnek kell változni, tehát a belső ciklust most a sorindex vezérli. A program tehát így alakul:

```
10 REM **** 67. PROGRAM ****
20 REM MATRIX OLVASASA OSZLOPFOLYTONOSAN
30 :
40 :
50 DIM MA(2,3)
60 FOR J=1 TO 3
70 FOR I=1 TO 2
80 PRINT J;"OSZLOP",I;"ELEM KÖVETKEZIK";
90 INPUT MA(I,J)
100 NEXT I
110 NEXT J
```

Ha a program írásakor nem tudjuk előre a mátrix méreteit, akkor a program használatakor ezeket is be kell olvasni, még a DIM utasítás végrehajtása előtt.

Zárópéldánk legyen egy általánosan használható mátrixbeolvasó szubrutin:

```
1000 REM **** 68. PROGRAM ****
1010 REM MATRIX BEOLVASAS
1020 :
1030 :
1040 INPUT "SOROK SZÁMA";N
1050 INPUT "OSZLOPOK SZÁMA";M
1060 IF N<=0 OR M<=0 THEN 1040
1070 DIM MA(N,M)
1080 FOR I=1 TO N
1090 FOR J=1 TO M
1100 PRINT I;"SOR",J;"ELEM";
1110 INPUT MA(I,J)
1120 NEXT J
1130 NEXT I
1140 RETURN
```

Az 1040-es és az 1050-es sorokban kérjük a sorok (N) és oszlopok (M) számát. Ha valamelyik érték véletlenül nulla vagy negatív szám, akkor újra olvassuk, amíg helyes értékeket nem kapunk. Ezután helyet foglalunk az MA mátrixnak, majd sorfolytonosan kérjük az elemeit.

Mi történik, ha ezt az alprogramot az öt használó főprogramból kétszer hívjuk meg?

A DIM utasítás *másodszori* végrehajtásakor fellép a

? REDIM'D ARRAY ERROR

hibaüzenet.

Arról, hogy hogyan kell olyan alprogramot írni, amely többször is hívható, a 4. fejezetben lesz szó.

Vegyük észre, hogy mátrix 0. sorát és 0. oszlopát nem olvasta be az alprogram. Természetesen a mátrix 0. sora és oszlopa akkor is helyet foglal a tárban, ha semmire sem használjuk.

A program érthetősége, áttekinthetősége céljából mégis célszerűbb a sor- és oszlop-indexeket a matematikából is megszokott módon 1-től N-ig, ill. 1-től M-ig használni, a 0. sort és oszlopot pedig felhasználhatjuk valami másra, pl. a sor- és oszlop-összegek tárolására.

2.5.2 Mátrixok kiírása

Egy mátrix elemeinek értékeit is kiírhatjuk két egymásba ágyazott ciklus segítségével.

A következő programrészlet pl. kiírja az $N \times M$ méretű MA mátrix elemeit a képernyőre, az elemeket két tizedesjegyre kerekítve.

```
2000 REM **** 69. PROGRAM ****
2010 REM N*M MERETU MA MATRIX KIIRASA
2020 :
2030 :
2040 FOR I=1 TO N
2050 FOR J=1 TO M
2060 PRINT INT(MA(I,J)*100+0.5)/100;
2070 NEXT J
2080 PRINT
2090 NEXT I
```

A 2060-as PRINT utasításban szereplő pontosvesszőre (;) azért van szükség, hogy a mátrixelemeket a program sorban egymás után írja. A 2080-as PRINT utasítással érjük el, hogy a mátrix egy sorának kiírása után a program a képernyőn is új sort kezdjen írni.

A bemutatott egyszerű programrészlet sajnos nem minden esetben ír áttekinthető táblázatot a képernyőre. Ennek az az oka, hogy a kiírt számok között a PRINT utasítás mindig egy üres karaktert hagy. Ha a számok nem egyforma hosszúak, akkor a mátrix oszlopai nem kerülnek egymás alá.

Valamivel rendezettebb kiírási formátumot érhetünk el a TAB függvény használatával, ha az oszlopindexből előre kiszámítjuk, hogy a mátrixelemet az adott képernyősor hányadik pozíciójától kezdve kell írni.

A TAB függvényben a kiírás kívánt kezdőpozícióját adhatjuk meg. Ha pl. a mátrixelemek elférnek a képernyő hat pozícióján, így szervezhetjük a kiírást:

```
2000 REM **** 70. PROGRAM ****
2010 REM MATRIX KIIRAS TAB FUGGVENNYEL
2020 :
2030 :
2040 FOR I=1 TO N
2050 FOR J=1 TO M
2060 PRINT TAB((J-1)*6); INT(MA(I,J)*100+.5)/100;
2070 NEXT J
2080 PRINT
2090 NEXT I
```

Szebb kiírási képet kaphatunk, ha a mátrix elemeit a 41-es programmal nyomtatjuk ki, amelynek hívó utasítása: GOSUB 3000. Ezt a 71. programban mutatjuk be:

```
2000 REM **** 71. PROGRAM ****
2010 :
2020 :
2030 FOR I=1 TO N
2040 FOR J=1 TO M
2050 A=MA(I,J):GOSUB 3000:PRINT#1,A$;
2060 NEXT J
2070 PRINT#1
2080 NEXT I
2090 RETURN
```

A hívó főprogramban gondoskodni kell az 1-es file megnyitásáról, F, T, U\$, N\$, C\$ értékeinek beállításáról.

Ha az OPEN1,3 utasítással az 1-es file-hoz a képernyőt rendeljük hozzá, a szubrutin a képernyőre írja az MA mátrixot, míg OPEN1,4 után a sornyomtaróra.

2.5.3 Műveletek mátrixokkal

Ebben a fejezetben a mátrixműveletek BASIC alprogramjait készítjük el. Az alprogramok mindig feltételezik, hogy a hívó főprogram már elvégezte a szükséges helyfoglalásokat (DIM) és beolvasta a számításokhoz szükséges adatokat.

A fejezet végén egy teljes program szemlélteti az alprogramok használatát.

Mátrixok összeadása

$A \quad C = [c_{ij}]_{n \times m}$ mátrix az $A = [a_{ij}]_{n \times m}$

és a $B = [b_{ij}]_{n \times m}$ mátrixok összege, ha

$$c_{ij} = a_{ij} + b_{ij} \quad (i = 1, \dots, n, j = 1, \dots, m)$$

A mátrixösszeadást megvalósító BASIC programnak tehát sorra kell vennie az A és a B második minden elemét, és az azonos indexűek összegét el kell helyeznie a C mátrix megfelelő helyén. A megoldás lényege újra az a két egymásba ágyazott ciklus, amiről a mátrixok beolvasásánál már bőven esett szó. Mindössze a ciklusok közös magját kell kicserélni.

Az $N \times M$ méretű A és B mátrixokat tehát a következő alprogram segítségével adhatjuk össze:

```
500 REM **** 72. PROGRAM ****
510 REM C(N*M)=A(N*M)+B(N*M)
520 :
530 :
540 FOR I=1 TO N : REM SORINDEX
550 FOR J=1 TO M : REM OSZLOPINDEX
560 C(I,J)=A(I,J)+B(I,J) : REM CIKLUSMAG
570 NEXT J
580 NEXT I
590 RETURN
```


Figyeljük meg, hogy az $(i=1, \dots, n, j=1, \dots, m)$ szimbólumot a BASIC programban mindig két egymásba ágyazott ciklussal írhatjuk le. A ciklusok magjába pedig az a művelet kerül, amit az összes mátrixelemen végre akarunk hajtani.

Ezután a felismerés után már könnyen megoldható egy *mátrix szorzása skalárral*. Az $A=[a_{ij}]_{n \times m}$ mátrixot úgy kell megszorozni a c skalárral, hogy a mátrix minden elemét megszorozzuk c -vel:

$$cA = [ca_{ij}]_{n \times m}.$$

A $c \cdot A$ elemei tehát:

$$(ca_{ij}) \quad (i=1, \dots, n, j=1, \dots, m)$$

A szorzást elvégző alprogram pedig:

```
600 REM **** 73. PROGRAM ****
610 REM A(N*M)=C*A(N*M)
620 :
630 :
640 FOR I=1 TO N
650 FOR J=1 TO M
660 A(I,J)=C*A(I,J)
670 NEXT J
680 NEXT I
690 RETURN
```

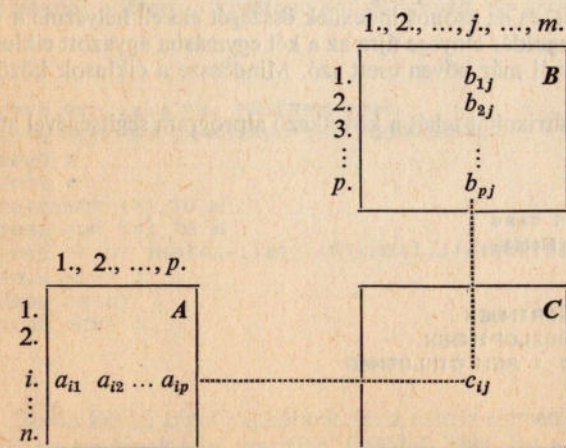
Mátrixok szorzása

Adott A és B mátrix csak akkor szorozható össze, ha A -nak ugyanannyi oszlopa van, mint ahány sora van B -nek.

$$A = [a_{ij}]_{n \times p} \quad B = [b_{ij}]_{p \times m}$$

$$A \cdot B = C[c_{ij}]_{n \times m}$$

A szorzatmátrix c_{ij} eleme az A mátrix i -edik sorvektorának és a B mátrix j -edik oszlopvektorának skalárszorzata.



$$c_{ij} = \sum_{l=1}^p a_{il} \cdot b_{lj} \quad (i=1, \dots, n, j=1, \dots, m)$$

A mátrixok szorzását megvalósító BASIC alprogramban a már megszokott két ciklus most egy harmadik, összegző ciklust is közre fog. A változó, ahová az összegnek kerülnie kell: C(I,J)

```
700 REM **** 74. PROGRAM ****
710 REM MÁTRIXOK SZORZÁSA
720 REM C(N*M)=A(N*P)*B(P*M)
730 :
740 :
750 FOR I=1 TO N
760 FOR J=1 TO M
770 S=0
780 FOR L=1 TO P
790 S=S+A(I,L)*B(L,J)
800 NEXT L
810 C(I,J)=S
820 NEXT J
830 NEXT I
840 RETURN
```

Mátrix hatványozása

Hatványozni csak négyzetes mátrixot lehet.

Az

$A = [a_{ij}]_{n \times n}$ mátrix k -adik hatványát a következőképpen kell kiszámítani:

$$B = A^k = A \cdot A \cdot A \dots A$$

1, 2, 3, ..., k -adik tényező.

$$A^0 = E_{n \times n}: \text{egységmátrix}$$

A hatványozást végző alprogram úgy működik, hogy először elhelyezi B-ben az egységmátrixot. Ha K értéke nulla, be is fejezi működését, hiszen B-ben éppen az A mátrix nulladik hatványa van. Ha K értéke nagyobb nullánál, a B mátrixot K-szor meg kell szorozni az A mátrixszal, de úgy, hogy a szorzás eredménye mindig a B mátrixba kerüljön vissza. A hatványozás minden ciklusában B új elemeit a következő algoritlussal számítjuk ki:

$$b_{ij} = \sum_{l=1}^n b_{il} \cdot a_{lj} \quad (i=1, \dots, n, j=1, \dots, n)$$

Az itt következő alprogrammal látszólag megoldhatjuk az A mátrix k -adik hatványának kiszámítását:


```

850 REM **** 75. PROGRAM ****
860 REM B(N*N)=A(N*N)↑K
870 :
880 :
890 REM B(N*N)-BE ELOSZOR AZ EGYSEGMAATRIX KERUL
900 :
910 FOR I=1 TO N
920 FOR J=1 TO N
930 B(I,J)=0 : IF I=J THEN B(I,J)=1
940 NEXT J
950 NEXT I
960 :
970 IF K=0 THEN 1130
980 :
990 REM ITT KEZDODIK A HATVANYOZAS
1000 REM A B(N*N)=B(N*N)*A(N*N)
1010 REM SZORZAST K-SZOR KELL ELVEGEZNI
1020 :
1030 FOR I1=1 TO K
1040 FOR I=1 TO N
1050 FOR J=1 TO N
1060 B(I,J)=0
1070 FOR L=1 TO N
1080 B(I,J)=B(I,J)+B(I,L)*A(L,J)
1090 NEXT L
1100 NEXT J
1110 NEXT I
1120 NEXT I1
1130 RETURN

```

Próbáljuk meg az alprogram segítségével kiszámítani az

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \text{ mátrix első hatványát!}$$

Ezt megtehetjük pl. a következő főprogrammal:

```

10 REM **** 76. PROGRAM ****
20 :
30 K=1 : N=2
40 DIM A(2,2), B(2,2)
50 :
60 A(1,1)=2 : A(1,2)=1
70 A(2,1)=1 : A(2,2)=1
80 :
90 GOSUB 850
100 :
110 PRINT B(1,1), B(1,2)
120 PRINT B(2,1), B(2,2)
130 END

```

Eredményül természetesen a

$$B = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \text{ mátrix kiírását várjuk, de meglepetésünkre a képernyőn a}$$

$$B = \begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} \text{ mátrix jelenik meg.}$$

A hiba okának kinyomozásához kövessük nyomon az alprogram működését! A 910—950-es sorokban valóban bekerül B-be az egységmátrix. (Erről meggyőződhetünk pl. úgy, hogy a főprogram 30-as sorában átírjuk a K=1 utasítást K=0-ra.) De hiába írtuk B-be az egységmátrixot, ha a 1060-as programsorban mindjárt el is rontjuk!

A ciklusbeli összegzés miatt indokolt a leendő összeget tartalmazó változó nullázása, de egyúttal tönkretettük a szorzandó mátrixot is!

A hibát úgy javíthatjuk ki, hogy az összegzéshez külön változót használunk és az összegző ciklus befejezése után helyezzük el a változó tartalmát a B mátrixban. Javítsuk tehát ki az alprogramot a következőképpen:

```

1055 REM *** 77. PROGRAM ***
1056 REM A 76. PROGRAM JAVITASA
1060 S=0
1070 FOR L=1 TO N
1080 S=S+B(I,L)*A(L,J)
1090 NEXT L
1095 B(I,J)=S

```

A javított program futásának eredménye még mindig kínosan érinti a program készítőjét, mert most pedig a

$$B = \begin{bmatrix} 2 & 2 \\ 1 & 2 \end{bmatrix} \text{ eredményt láthatja a képernyőn.}$$

Az első oszlopban már a várakozásunknak megfelelő számok szerepelnek, de mi történt a második oszloppal?

A rutinnak a következő mátrixok összeszorozását kellett volna elvégeznie:

$$\hat{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

$B \quad A$

$B(1,1)=1 \cdot 2 + 0 \cdot 1 = 2$ kiszámítása helyes. De mivel az eredményt rögtön el is helyeztük B(1,1)-ben, B(1,2) kiszámításához már ezt az új értéket használta programunk.

A szorzatmátrix második sorának számításánál ugyanez történt, a

$$B(2,1)=B(2,1) \cdot A(1,1)+B(2,2) \cdot A(2,1)=0 \cdot 2+1 \cdot 1=1$$

még jó, de a

$$B(2,2)=B(2,1) \cdot A(1,2)+B(2,2) \cdot A(2,2)=1 \cdot 1+1 \cdot 1=2$$

már hibás, hiszen B(2,1) értéke megváltozott.

A hibát tehát azzal követtük el, hogy B elemeit túlságosan hamar változtattuk meg. Mindaddig szükségünk van az i -edik sorban szereplő régi elemekre, amíg a sor új elemeinek számítása be nem fejeződik. A rutin javításához tehát szükség van egy segédvektorra, amelyben tárolhatjuk a szorzatmátrix i -edik sorát addig, amíg annak az összes elemét ki nem számítottuk. Ha végeztünk az i -edik sor számításával, ebből a segédvektorból most már átvihetjük az elemeket a B mátrix i -edik sorába, hiszen az itt szereplő régi értékekre nincs többé szükségünk. Segédvektorként felhasználhatjuk például a B mátrix nulladik sorát.

A most már tökéletesen működő alprogramunk a javítások elvégzése után:

```
850 REM **** 73. PROGRAM ****
860 REM B(N*N)=A(N*N)*K
870 :
880 :
890 REM B(N*N)-BE ELOSZOR AZ EGYSEGMAATRIX KERUL
900 :
910 FOR I=1 TO N
920 FOR J=1 TO N
930 B(I,J)=0 : IF I=J THEN B(I,J)=1
940 NEXT J
950 NEXT I
960 :
970 IF K=0 THEN 1130
980 :
990 REM ITT KEZDODIK A HATVANYOZAS
1000 REM A B(N*N)=B(N*N)*A(N*N)
1010 REM SZORZAST K-SZOR KELL ELVEGEZNI
1020 :
1030 FOR I1=1 TO K
1040 FOR I=1 TO N
1050 FOR J=1 TO N
1060 S=0
1070 FOR L=1 TO N
1080 S=S+B(I,L)*A(L,J)
1090 NEXT L
1095 B(I,J)=S
1100 NEXT J
1101 :
1102 REM AZ I-EDIK SOR SZAMITASA BEFEJEZVE
1103 REM A SOR A HELYERE KERUL
1104 :
1105 FOR L=1 TO N
1106 B(I,L)=B(0,L)
1107 NEXT L
1108 :
1110 NEXT I
1120 NEXT I1
1130 RETURN
```

Mátrixnormák

Az A mátrix $\|A\|$ normája egy valós szám, amely eleget tesz a következő feltételeknek:

- $\|A\| \geq 0$, és $\|A\| = 0$ akkor és csak akkor, ha $a_{ij} = 0$, $(i = 1, \dots, n, j = 1, \dots, n)$
- $\|cA\| = |c| \cdot \|A\|$ (c konstans)
- $\|A+B\| \leq \|A\| + \|B\|$ ha $A+B$, ill.
- $\|A \cdot B\| \leq \|A\| \cdot \|B\|$ $A \cdot B$ létezik.

Az $A=[a_{ij}]_{n \times m}$ mátrix leggyakrabban használt normái a következők:

$$\|A\|_1 = \max_i \sum_{j=1}^m |a_{ij}|$$

$$\|A\|_2 = \max_j \sum_{i=1}^n |a_{ij}|$$

$$\|A\|_3 = \sqrt{\sum_{i=1}^n \sum_{j=1}^m |a_{ij}|^2}$$

Készítsünk alprogramot az $A=[a_{ij}]_{n \times m}$ mátrix első normájának kiszámítására! Az $\|A\|_1$ meghatározásához először képeznünk kell az

$$\hat{a}_i = \sum_{j=1}^m |a_{ij}| \quad (i = 1, \dots, n)$$

összegeket, majd ezek közül ki kell választanunk a legnagyobbat.

Az $\hat{a}_i$ értékek tárolásához szükségünk van egy n elemű vektorra, használhatjuk erre a célra pl. az A mátrix θ -edik oszlopát.

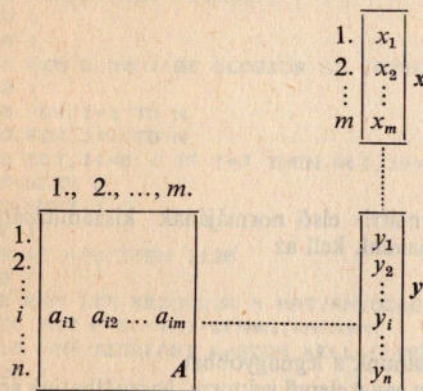
```
1200 REM **** 79. PROGRAM ****
1210 REM N1---A(N,M) ELSO NORMAJA
1220 :
1230 :
1240 FOR I=1 TO N
1250 :
1260 REM AZ I-EDIK SOR OSSZEGE A(I,0)-BA
1270 REM KERUL
1280 :
1290 S=0
1300 FOR J=1 TO M
1310 S=S+ABS(A(I,J))
1320 NEXT J
1330 A(I,0)=S
1340 :
1350 NEXT I
1360 :
1370 REM 0-ADIK OSZLOP LEONAGYOBB ELEMENEK
1380 REM KERESESE
1390 :
1400 N1=A(1,0)
1410 FOR I=2 TO N
1420 IF A(I,0)>N1 THEN N1=A(I,0)
1430 NEXT I
1440 :
1450 RETURN
```


Mátrix szorzása vektorral

A mátrixok szorzásának gyakran előforduló speciális esete, amikor a jobb oldali mátrixtényezőnek (szorzó mátrixnak) csak egyetlen oszlopa van. Ebben az esetben a szorzatmátrixnak is csupán egy oszlopa lesz.

A mátrixok szorzásánál használt rajz segítségével ezt az esetet így szemléltethetjük, ha

$$A = [a_{ij}]_{n \times m}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \text{és} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$



$$y = Ax, \quad y_i = \sum_{j=1}^m a_{ij} \cdot x_j \quad (i = 1, \dots, n)$$

A szorzást megvalósító alprogram tehát a következő:

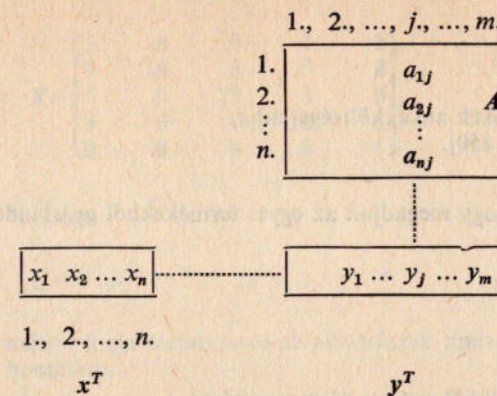
```
1500 REM **** 80. PROGRAM ****
1510 REM Y[N]=A[N,M]*X[M]
1520 :
1530 :
1540 FOR I=1 TO N
1550 S=0
1560 FOR J=1 TO M
1570 S=S+A(I,J)*X(J)
1580 NEXT J
1590 Y(I)=S
1600 NEXT I
1610 RETURN
```

Másik speciális eset, amikor a bal oldali mátrixtényező egyetlen sorból áll. Ilyenkor a szorzatmátrixnak is egyetlen sora van, vagyis eredményül sorvektort kapunk. Bár programozástechnikailag teljesen közömbös, hogy egy vektor sor- vagy oszlopvektor-e, az összhang kedvéért az x sorvektor jelölésére itt is megtartjuk a matematikában használatos x^T jelölést.

Feladatunk most tehát a következő szorzat kiszámítása:

$$y^T = x^T \cdot A, \quad \text{ahol} \quad A = [a_{ij}]_{n \times m}, \quad x^T = [x_1, \dots, x_n],$$

$Y^T[y_1, \dots, y_m] \cdot Y^T$ értékeinek kiszámítását leolvashatjuk a már ismert rajzról.



$$y_j = \sum_{i=1}^n a_{ij} \cdot x_i \quad (j = 1, \dots, m)$$

A szorzást elvégző alprogram:

```
1650 REM **** 81. PROGRAM ****
1660 REM Y[M]=X[N]*A[N,M]
1670 :
1680 :
1690 FOR J=1 TO M
1700 S=0
1710 FOR I=1 TO N
1720 S=S+X(I)*A(I,J)
1730 NEXT I
1740 Y(J)=S
1750 NEXT J
1760 RETURN
```

A gazdasági számításokat gyakran megfogalmazhatjuk mátrixműveletekkel.

Tegyük fel pl. hogy egy üzem m féle terméket gyárt n féle alkatrészből. Az egyes termékekhez szükséges alkatrészek jegyzékét mátrixszal adhatjuk meg.

Ha a mátrix oszlopai a terméktípust, sorai pedig az alkatrésztípust jelentik, akkor az a_{ij} elem megmondja, hogy a j -edik termékhez hány darabra van szükség az i -edik alkatrészből.

Pl. az

$$A = \begin{bmatrix} 3 & 2 & 1 & 0 & 4 \\ 5 & 6 & 0 & 7 & 2 \\ 4 & 5 & 3 & 2 & 1 \end{bmatrix}$$

mátrix öt termék előállítását fejezi ki három féle alkatrészből. Az $a_{14}=0$ azt jelenti, hogy a negyedik termék előállításához nincs szükség az első alkatrésze.
Adjuk meg az egyes alkatrészek árait egy K költségvektorban:

$K^T = [100, 200, 50]$, $K_1 = 100$ azt jelenti, hogy az első alkatrészből egy darab 100 Ft-ba kerül.

Az

$Y^T = K^T \cdot A$ szorzat az egyes termékek anyagköltségét adja.
 $Y^T = [1500, 1650, 250, 1500, 850]$.

Bővíthetjük a feladatot pl. úgy, hogy megadjuk az egyes termékekből gyártandó darabszámokat.

Legyen

$$x = \begin{bmatrix} 5 \\ 10 \\ 20 \\ 100 \\ 1 \end{bmatrix}, x_j \text{ jelenti a } j\text{-edik termékből gyártandó mennyiséget.}$$

A $z = A \cdot x$ vektor elemeiből leolvashatjuk, hogy a kívánt termékszámok gyártásához melyik alkatrészből mennyire van szükség.

$$z = \begin{bmatrix} 59 \\ 787 \\ 331 \end{bmatrix}$$

Az $a = K^T \cdot A \cdot x = K^T \cdot z$ mennyiség pedig a teljes anyagköltséget adja.

Példánkban:

$$a = 59 \cdot 100 + 787 \cdot 200 + 331 \cdot 50 = 179850 \text{ Ft.}$$

A költségvektort diagonális mátrix formájában is írhatjuk:

$$K = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

Ekkor az

$M = K \cdot A$ mátrix az egyes termékek anyagköltségét tartalmazza alkatrészenkénti felbontásban.

$$M = \begin{bmatrix} 300 & 200 & 100 & 0 & 400 \\ 1000 & 1200 & 0 & 1400 & 400 \\ 200 & 250 & 150 & 100 & 50 \end{bmatrix}$$

Ha diagonális mátrixokba írjuk az egyes termékekből gyártandó darabszámokat,

$$X = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

akkor a

$$Q = A \cdot X$$

mátrix fogja tartalmazni az alkatrészek típusonkénti számát termékek szerinti felbontásban.

$$Q = \begin{bmatrix} 15 & 20 & 20 & 0 & 4 \\ 25 & 60 & 0 & 700 & 2 \\ 20 & 50 & 60 & 200 & 1 \end{bmatrix}$$

Készítsünk BASIC programot az előbbi feladat megoldására a rendelkezésünkre álló alprogram-gyűjtemény felhasználásával!

A program fő lépései

1. Változók kezdőértékének beállítása a hívott rutinok számára.
2. Nyomtató file megnyitása.
3. Adatok beolvasása
 - alkatrészek száma (N),
 - termékek száma (M),
 - alkatrészejegyzék ($A = [a_{ij}]_{n \times m}$),
 - alkatrészek árai ($k_1, \dots, k_n$),
 - gyártandó darabszám ($x_1, \dots, x_m$).
4. Alapadatok nyomtatása.
5. $y^T = K^T \cdot A$, anyagköltségek számítása.
6. ($y_1, \dots, y_m$) nyomtatása.
7. $z = A \cdot x$, alkatrész-szükséglet számítása.
8. ($z_1, \dots, z_n$) nyomtatása.
9. $a = K^T \cdot x$ anyagköltség számítása, nyomtatása.


```

10 REM **** 82. PROGRAM ****
20 REM ANYAGKOLTSEGSZAMITAS
30 :
40 :
50 REM VALTOZOK BEALLITASA
60 :
70 SP$=" "
80 C$="*****"
90 Z$="000"
100 :
110 REM NYOMTATO FILE MEGNYITASA
120 :
130 INPUT "KIIRAS KESPERNYORE/ANYOMTATORA";Q$
140 Q=3:IF Q$="N" THEN Q=4
150 OPEN 1,Q
160 :
170 REM ADATOK BEOLVASASA
180 :
190 INPUT "ALKATRESZEK SZAMA";N
200 INPUT "TERMEKEK SZAMA";M
210 DIM A(N,M),K(N),X(M),Y(M),Z(N)
220 FOR J=1 TO M
230 PRINT "J";J;". TERMEK ALKATRESZIGENYE"
240 FOR I=1 TO N
250 PRINT I;". ALKATRESZBOL";
260 INPUT A(I,J)
270 NEXT I
280 NEXT J
290 PRINT "ALKATRESZ ARAK"
300 FOR I=1 TO N
310 PRINT I;". ALKATRESZ (FT)";
320 INPUT K(I)
330 NEXT I
340 PRINT "GYARTANDO DARABSZAMOK"
350 FOR I=1 TO M
360 PRINT I;". TERMEK (DB)";
370 INPUT X(I)
380 NEXT I
390 :
400 :
410 REM ADATOK NYOMTATASA
420 PRINT#1,"ALKATRESZFELHASZNALAS TERMEKENKENT"
430 GOSUB 1410
440 V$="":GOSUB 1840
450 PRINT#1,"ALKATRESZ ARAK"
460 P=9:S=2
470 FOR I=1 TO N
480 A=K(I):GOSUB 1600
490 PRINT#1,"A";I;": ";A$;" FT"
500 NEXT I
510 GOSUB 1840
520 PRINT#1,"GYARTANDO DARABSZAMOK"
530 P=11:S=0
540 FOR I=1 TO M
550 A=X(I):GOSUB 1600
560 PRINT#1,"T";I;": ";A$;" DB"
570 NEXT I
580 GOSUB 1840
590 :

```

```

600 REM ANYAGKOLTSEGEK SZAMITASA.NYOMTATASA
610 GOSUB 1150 : REM Y(M)=K(N)*A(N,M)
620 PRINT#1,"ANYAGKOLTSEGEK TERMEKENKENT"
630 P=9:S=2
640 FOR I=1 TO M
650 A=Y(I):GOSUB 1600
660 PRINT#1,"T";I;": ";A$;" FT"
670 NEXT I
680 GOSUB 1840
690 :
700 REM ANYAGSZUKSEGLET SZAMITASA
710 :
720 GOSUB 1000 : REM Z(N)=A(N,M)*X(M)
730 PRINT#1,"SZUKSEGES ALKATRESZEK SZAMA"
740 P=11:S=2
750 FOR I=1 TO N
760 A=Z(I):GOSUB 1600
770 PRINT#1,"A";I;": ";A$;" DB"
780 NEXT I
790 GOSUB 1840
800 :
810 REM TELJES ANYAGKOLTSEG
820 :
830 GOSUB 1300 : REM A=K(N)*Z(N)
840 PRINT#1,"ANYAGKOLTSEG OSSZESEN";A;" FT"
850 V$="":GOSUB 1840
860 CLOSE 1
870 END
1000 REM **** 80. PROGRAM ****
1010 REM Z(N)=A(N,M)*X(M)
1020 :
1030 :
1040 FOR I=1 TO N
1050 S=0
1060 FOR J=1 TO M
1070 S=S+A(I,J)*X(J)
1080 NEXT J
1090 Z(I)=S
1100 NEXT I
1110 RETURN
1120 :
1130 :
1140 :
1150 REM **** 81. PROGRAM ****
1160 REM Y(M)=K(N)*A(N,M)
1170 :
1180 :
1190 FOR J=1 TO M
1200 S=0
1210 FOR I=1 TO N
1220 S=S+K(I)*A(I,J)
1230 NEXT I
1240 Y(J)=S
1250 NEXT J
1260 RETURN
1270 :

```



```

1300 REM A=K(I)*Z(I)
1310 :
1320 :
1330 A=0
1340 FOR I=1 TO N
1350 A=A+K(I)*Z(I)
1360 NEXT I
1370 RETURN
1380 :
1390 :
1400 :
1410 REM **** 74. PROGRAM ****
1420 :
1430 :
1440 P=9:S=2
1442 PRINT#1," ";
1450 FOR J=1 TO M
1460 PRINT#1," T";J;
1470 NEXT J
1480 V$="":GOSUB 1840
1490 FOR I=1 TO N
1500 PRINT#1,"A";I;": ";
1510 FOR J=1 TO M
1520 A=A(I,J):GOSUB 1600:PRINT#1,A;
1530 NEXTJ
1540 PRINT#1
1550 NEXTI
1560 RETURN
1570 :
1580 :
1590 :
1600 REM ****          ****
1610 REM AZ A SZAM KONVERTALASA KIIRASHOZ
1620 :
1630 REM A$-----EREDMENY
1640 REM P-----A SZAM HOSSZA
1650 REM S-----TIZEDESEK SZAMA
1660 :
1670 :
1680 L=P-S-2 : REM EGESZ JEGYEK SZAMA
1690 A3$="":IFA(0)THEN A=-A:A3$="-"
1700 A1=INT(A) : REM A EGESZ RESZE
1710 IF ABS(A1)>101L-1 OR L<1 THEN GOTO1810
1720 IF S=0 THEN A2$="":GOTO 1770
1730 A2=A-A1 :REM A TORT RESZE
1740 A2=INT(A2*101S+0.5)
1750 A2$=STR$(A2):A4=LEN(A2$)-1
1760 A2$=LEFT$(A2$,S-A4)+RIGHT$(A2$,A4)+2$
1770 A1$=STR$(A1):A1=LEN(A1$)
1780 A1$=SP$+A3$+RIGHT$(A1$,A1-1)
1790 A$=RIGHT$(A1$,L+1)+". "+LEFT$(A2$,S)
1800 RETURN
1810 A$=LEFT$(C$,P)
1820 RETURN
1830 :
1840 REM ELVALASZTO $OR

```

```

1845 :
1850 PRINT#1
1860 FOR L=1 TO 60
1870 PRINT#1,V$;
1880 NEXT L
1890 PRINT#1:PRINT#1
1900 RETURN

```

A számításokhoz használt alprogramok

1150—1260 Mátrix szorzása sorvektorral balról.
 1000—1111 Mátrix szorzása oszlopvektorral jobbról.
 1300—1370 Két vektor skaláris szorzata.

A kiíráshoz használt alprogramok

1410—1560 Az A mátrix kiírását végző rutin csak annyiban különbözik a 71. programtól, hogy a mátrix sorait és oszlopait sorszámaikkal együtt írja ki.
 1600—1800 Számkonvertáló rutin.
 1840—1900 Elválasztó sor nyomtatása.
 Az eredménytáblázat tagolására használjuk. A nyomtatni kívánt elválasztó karaktert a rutin hívása előtt a V\$ változóban kell elhelyezni.

A 82. program futtatásának eredménye:

```

ALKATRESZFELHASZNALAS TERMEKENKENT
      T 1      T 2      T 3      T 4      T 5
-----
A 1 :    +3.00    +2.00    +1.00    +0.00    +4.00
A 2 :    +5.00    +6.00    +0.00    +7.00    +2.00
A 3 :    +4.00    +5.00    +3.00    +2.00    +1.00

=====

ALKATRESZ ARAK
A 1 :   +100.00 FT
A 2 :   +200.00 FT
A 3 :   +50.00 FT

=====

GYARTANDO DARABSZAMOK
T 1 :      +5. DB
T 2 :     +10. DB
T 3 :     +20. DB
T 4 :    +100. DB
T 5 :      +1. DB

=====

ANYAGKOLTSEGEK TERMEKENKENT
T 1 :  +1500.00 FT
T 2 :  +1650.00 FT
T 3 :   +250.00 FT
T 4 :  +1500.00 FT
T 5 :   +850.00 FT

=====

SZUKSEGES ALKATRESZEK SZAMA
A 1 :     +59.00 DB
A 2 :   +787.00 DB
A 3 :  +331.00 DB

=====

ANYAGKOLTSEG OSSZESEN 179850 FT

*****

```

3. FEJEZET

Programozástechnikai eszközök

Egy feladat megoldására nagyon sokféle program készíthető. Minden programozónak megvan a maga sajátos stílusa, s így azonos feladatra írt, megegyező tagolású programok is jelentősen különbözhetnek egymástól.

Egy program végső formáját objektív és szubjektív tényezők befolyásolják. Objektív tényezők pl. a programfejlesztéshez használt gép lehetőségei, a programfejlesztéshez rendelkezésre álló idő.

Szubjektív tényező a program készítőjének, (készítőinek) egyénisége, gyakorlata, rátermettsége.

A jó programot lehetőség szerint jellemzik a következő tulajdonságok:

- pontos;
- érthető;
- számítástechnikailag hatékony;
- gépi vagy nyelvi specialitásoktól független;
- jól dokumentált.

A program *pontossága* alatt azt értjük, hogy a számára kijelölt feladatot jól oldja meg, mégpedig minden — a program által elfogadott — bemenő adathalmazra.

A program *érthetősége* azt jelenti, hogy a programlista jól olvasható, a programrészek elkülönülnek egymástól, a fontos részek magyarázata szerepel a programlistában. Növeli a program érthetőségét, ha változóinak nevei utalnak a változó tartalmára. Célszerű a fontosabb változók jelentését magyarázó sorokban rögzíteni. Kerüljük a bonyolult megoldásokat még akkor is, ha ezzel néhány másodpercenyi futásidőt takaríthatunk meg.

A *számítástechnikailag hatékony* program nem pazarolja a tárat és a gépidőt. A programok tárigény és futási idő szerinti optimalizálását egyidejűleg nem lehet minden esetben megoldani, de a programok hatékonyságát akár az egyik, akár a másik szempont szerint gyakran igen egyszerű módszerekkel javíthatjuk.

A magyarországi számítógéppark igen sokféle gépből áll, és ráadásul meglehetősen gyorsan cserélődik. Aki tehát szeretne jól használható, sokoldalú programot készíteni, jól teszi, ha *függetleníti* magát annak a *gépnek a specialitásaitól*, amin éppen dolgozik.

Bosszantó hiba, ha pl. egy-egy nagyon szellemes programrendszer kizárólag egy speciális, ráadásul elavult nyomtatóra tudja kiírni az eredménylistákat. Sajnos előfordulhat, hogy kénytelenek vagyunk kihasználni a gépi adottságokat, vagy azért, mert az adott probléma másképpen nem oldható meg, vagy azért, mert ez a megoldás nagy mértékben növeli a program hatékonyságát. Az ilyen esetekben tanácsos a programnak a gépi specialitásokat kihasználó részét önálló alprogramként megírni, hogy a program más gépi környezetbe kerülve könnyen módosítható legyen.

Egyetlen programozási munka sem tekinthető befejezettnek, amíg nincs *dokumentálva*. A programokhoz feltétlenül készíteni kell egy *kezelési útmutatót*. A kezelési útmutató a program használója számára készül, és a program használatához szükséges

tudnivalókat tartalmazza. El kell készíteni ezen kívül a *program dokumentációját* is. A dokumentációt a programozó elsősorban saját maga és azok számára írja, akiknek esetleg egyszer a programot továbbfejleszteniük vagy módosítaniuk kell.

Ebben a fejezetben leírjuk azt a folyamatot, amelynek végterméke a felsorolt kíválmaknak megfelelő program, és bemutatunk néhány fogást, amellyel a programok hatékonysága növelhető.

3.1 PROGRAMTERVEZÉS

A programkészítés első fázisa a *probléma elemzése*. Az elemzés során feltétlenül tisztáznunk kell a leendő program feladatát. Ha már tisztában vagyunk azzal, hogy a program milyen célra készül, gyűjtsük össze a feladat kritikus pontjait. Nézzünk utána, hogy készítettünk-e már olyan programot, amelyben hasonló problémák merültek fel. Érdemes elgondolkozni azon is, hogy az eredeti ötlet általánosabbá tehető-e anélkül, hogy a feladat túlságosan bonyolulttá válna. A feladat elemzése során még nem kell jelentőséget tulajdonítanunk annak, hogy hogyan fogalmazzuk majd meg a programot a gép számára érthető nyelven, de mindenképpen tisztáznunk kell, hogy a rendelkezésünkre álló gépen a kiszemelt feladat egyáltalán megoldható-e.

A tervezésnek ez a szakasza anyaggyűjtés, amelynek eredménye a program céljának pontos megfogalmazása, és a megoldásra leginkább alkalmas módszer kiválasztása.

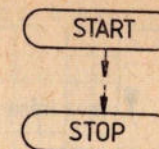
A tervezés következő szakaszában bontsuk részekre a feladatot! A részeket elemezzük külön-külön! Készítsünk vázlatot a szükséges programokról és alprogramokról anélkül, hogy kitérnénk a programozástechnikai részletekre! A vázlatokban mindig válaszoljunk a következő kérdésekre:

- mi a programrész feladata;
- milyen adatokra van szüksége;
- milyen eredményeket szolgáltat?

Amennyiben a programrész valamilyen számítást végez, írjuk le a számítás algoritmusát!

A programrészek kapcsolatának leírásához hasznos segédeszköz a folyamatábra, amellyel jól szemléltethető a készülő program szerkezete. A folyamatábra elemei az 5., a 6. és 7. ábrán láthatók.

Program eleje:

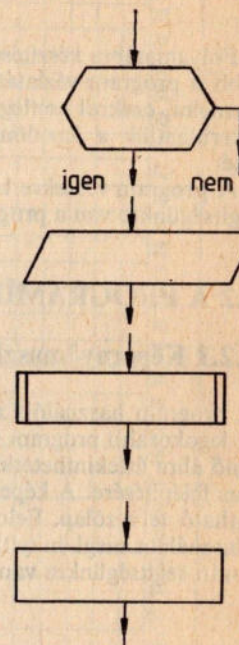


Program vége (végei):

Elágazás szimbólum:

A blokk belsejében egy eldöntendő kérdés kerül. A kérdésre adott válasz szerint a programnak más-más tevékenységet kell végrehajtania, ezen a ponton a program elágazik.

Bevitel — kivitel művelete:



Alprogram hívása:

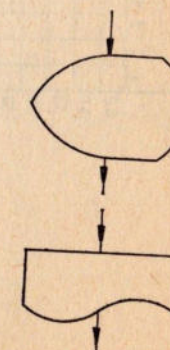
Értékkadás:

5. ábra A folyamatábra alapelemei

A bevitel — kivitel általános szimbóluma helyett használhatunk olyan jelöléseket is, amelyekről látszik, hogy a műveletet milyen készüléken akarjuk végezni:

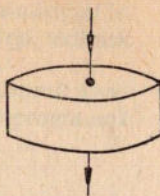
Bevitel billentyűzetről:

Kiírás nyomtatóra:



6. ábra A folyamatábra elemei

Lemez írása vagy olvasása:



7. ábra A folyamatábra elemei

Folyamatábra készítésénél nem érdemes minden apró részletet kidolgozni! Sokkal jobb a program vázlatát elkészíteni, és a durva vázlat alapján az egyes részleteket kiemelni, ezekről esetleg részletesebb, alárendelt ábrát rajzolni. Ezzel a módszerrel elkerülhetjük a lepedőnyi, áttekinthetetlen, használhatatlan folyamatábrák készítését.

A program részekre bontása nemcsak a programozási munkát könnyíti meg, de segítségünkre van a programok tesztelésénél, esetleges módosításánál is.

3.2 A PROGRAMÍRÁS KÖZVETLEN ELŐKÉSZÜLETEI

3.2.1 Képernyőmaszkok tervezése

A program használója a programból csak azt látja, ami a képernyőn megjelenik. A legokosabb program is csapnivaló fércmunkának látszik, ha a képernyőn megjelenő ábra áttekinthetetlen. Érdemes tehát gondot fordítani a képernyő világos, logikus felépítésére. A képernyőtervek készítésének jól bevált segédeszköze az 8. ábrán látható tervezőlap. Felosztása megfelel a képernyő 25×40-es mezőfelbontásának. Használata megkönnyíti a tetszetős képernyőmaszk kialakítását, és a programozás során segítségünkre van a kurzor pozicionálásában.

Képernyőmaszk tervezőlap Commodore 64-esre																																					
Jelölések:												Programnév:																									
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	31	32	33	34	35	36	37	38	39	40			

3.2.2 File-ok tervezése

Amennyiben a feladat elemzése során úgy döntöttünk, hogy bizonyos adathalmazokat lemezen vagy mágnesszalagon tárolunk, pontosan meg kell határoznunk az adathalmaz típusát, felépítését, a tárolt adatok szerkezetét. Adatfile-ok tervezéséről, használatáról bővebben beszélünk majd az 5.4. alfejezetben.

3.2.3 A nyomtatás tervezése

A program-előkészítéshez minden esetben hozzá tartozik a nyomtatásképek tervezése is. Ugyanúgy, mint a képernyőmaszkok tervezésénél, itt is fontosak az esztétikai szempontok, de talán még fontosabb annak eldöntése, hogy *mit* nyomtassunk ki.

Ha egy program túl sok listát készít, akkor senki sem fogja venni a fáradságot, hogy kihámozza belőle a lényeges információt.

A program készítése és tesztelése közben szükség lehet a részeredmények nyomtatására, de miután meggyőződünk róla, hogy programunk helyesen működik, ezeket a részeket feltétlenül hagyjuk ki belőle!

Az áttekinthető nyomtatásképp kialakítása nagyon körülményes és időt rabló munka még olyan nyelveken is, amelyek segítséget nyújtanak a változók formázott kiírásához. A C 64-es BASIC nyelve pedig alig ad támogatást a rendezett nyomtatásképp elkészítéséhez. Szükség van ezért néhány rutinra, amellyel megkönnyíthetjük magunknak ezt a fáradságos munkát. Ilyen rutinok találhatók a 2. fejezetben.

3.2.4 A változók elnevezése

Sajnos a C 64-es BASIC nem engedi meg a hosszú változónevek használatát. Némi fantáziával mégis adhatunk a változóinknak olyan nevet, amely utal a tartalomra. (Igaz ugyan, hogy a programlistában használhatunk két karakternél hosszabb változóneveket, de nehezen felderíthető hibát okozhat, ha pl. elfeledkezünk arról, hogy az AMIN és az AMAX változónevek ugyanazt a változót takarják.)

A programban azonos feladatra lehetőleg mindig ugyanazt a változót használjuk! Ezzel egyrészt sokkal áttekinthetőbbé válik a program, másrészt, ha programkönyvtárat alakítunk ki magunknak, akkor az azonos szellemben használt változónevek nem zavarják egymást a programok összefűzése során.

A változók típusának megválasztásakor gondoljunk pl. arra, hogy az egész típusú tömbök fele akkora tárhelyet foglalnak el, mint a valós típusúak, viszont az egész típusú skalár változók helyfoglalása ugyanakkora, mint a valós típusúaké, ráadásul a C 64-es egész számokkal lassabban számol, mint valósakkal.

Változóinkról készítsünk listát, amelyben leírjuk, hogy a programban melyik változó mit jelent!

3.2.5 A program dokumentációja

A programdokumentáció a következő részekből áll:

- a program célja és folyamatábrája;
- file-leírások;
- a használt algoritmusok leírása;
- nyomtatásképek;
- változólista;
- programlista, a programlista magyarázata.

Mint láttuk, a programdokumentáció nagyobb része már a programtervezés során készül, kivéve a programlistát és a hozzá csatolt magyarázatokat. Hosszú és bonyolult programok készítésekor a magyarázatot érdemes a programozási munkával párhuzamosan írni, mert néhány hét elteltével már nagyon nehéz a saját programlistánk megfejtése is.

3.2.6 Mintafeladat

Példaként az elmondottakra tervezzünk meg és készítsünk egy programot, mely egy benzinkútnál kialakuló állapot szimulációját végzi!

Szimulációnak azokat a rendszermodelleken végzett kísérleteket nevezzük, amelyeknek célja a rendszer viselkedésének meghatározása a modell viselkedésén keresztül. A szimuláció egy speciálisan számítógépes módszer arra, hogy előre lássuk a változtatások következményeit, anélkül, hogy azokat valóban végre kellene hajtani, így a rendszer nemkívánatos vonásait kiküszöbölhetjük, még mielőtt azt a gyakorlatban megvalósítanánk.

A *program célja*: egy benzinkútnál kialakuló állapot szimulációja a megadott eltételek mellett. A programmal közölni kell:

- a töltőhelyek számát;
- két kocs érkezése között eltelt idő várható értékét;
- a minimális és a maximális kiszolgálási időt;
- a megfigyelés időtartamát.

A program feltételezi, hogy a kocsik érkezésének valószínűsége minden percben azonos, és a kiszolgálási idő is egyenletes eloszlású véletlen változó. Ha minden kút foglalt, akkor annak a valószínűsége, hogy az utas hajlandó várakozni, fordítottan arányos a legrövidebb sorbeli kocsik számával. Amennyiben az utas úgy dönt, hogy marad, a legrövidebb sor végére áll.

A program egyperces időegységeket véve szimulálja a benzinkút forgalmát, a képernyőre kirajzolja a töltőhelyek előtt várakozó sorokat, számolja, hogy a megfigyelt időtartam alatt hány kocs fordult vissza, megjegyzi és kiírja, hogy a megfigyelés időszakában hány kocsiból állt a leghosszabb sor, óránkénti bontásban közli az áthaladó és visszaforduló kocsik számát, az átlagos várakozási időt.

A program kritikus része a véletlen események szimulálása. Erre a célra a C 64-es véletlenszám-generátorát használhatjuk fel, mely valós számok egyenletes eloszlású sorozatát adja a $[0, 1)$ intervallumon.

A bemenő adatokból tudjuk, hogy átlagosan mennyi idő telik el két kocs érkezése között. Legyen ez az érték a . Ekkor annak valószínűsége, hogy egy adott percben érkezik kocs a benzinkúthoz $1/a$. A programnak azon a pontján, ahol azt kell eldönteni, hogy az adott percben érkezett-e kocs a kúthoz, generálunk egy vélet-

len számot, ha ennek értéke kisebb mint $1/a_i$, akkor jött kocsi, ha nagyobb, akkor nem jött.

A feltételekből tudjuk, hogy annak a valószínűsége, hogy a kocsi itt is marad, fordítottan arányos a legrövidebb kocsisor hosszával, legyen ez MIN. Amennyiben a generált véletlen szám kisebb mint $1/\text{MIN}$, úgy döntünk, hogy a kocsi marad, ha nagyobb, akkor visszafordul.

Abban a percben, amikor egy kocsi tankolása megkezdődik, meg kell jósolnunk a kiszolgálási idő hosszát. A bemenő adatokból ismerjük a kiszolgálási idő minimális, és maximális értékét, legyenek ezek K1 és K2. A feltételeink szerint a kiszolgálási időtartam olyan egész szám, mely egyforma valószínűséggel veheti fel a K1 és K2 közötti értékek bármelyikét.

Ilyen számot az $r \in [0, 1]$ intervallumba eső véletlen szám segítségével így állhatunk elő:

$$K = [r(K2 - K1) + K1] + 1,$$

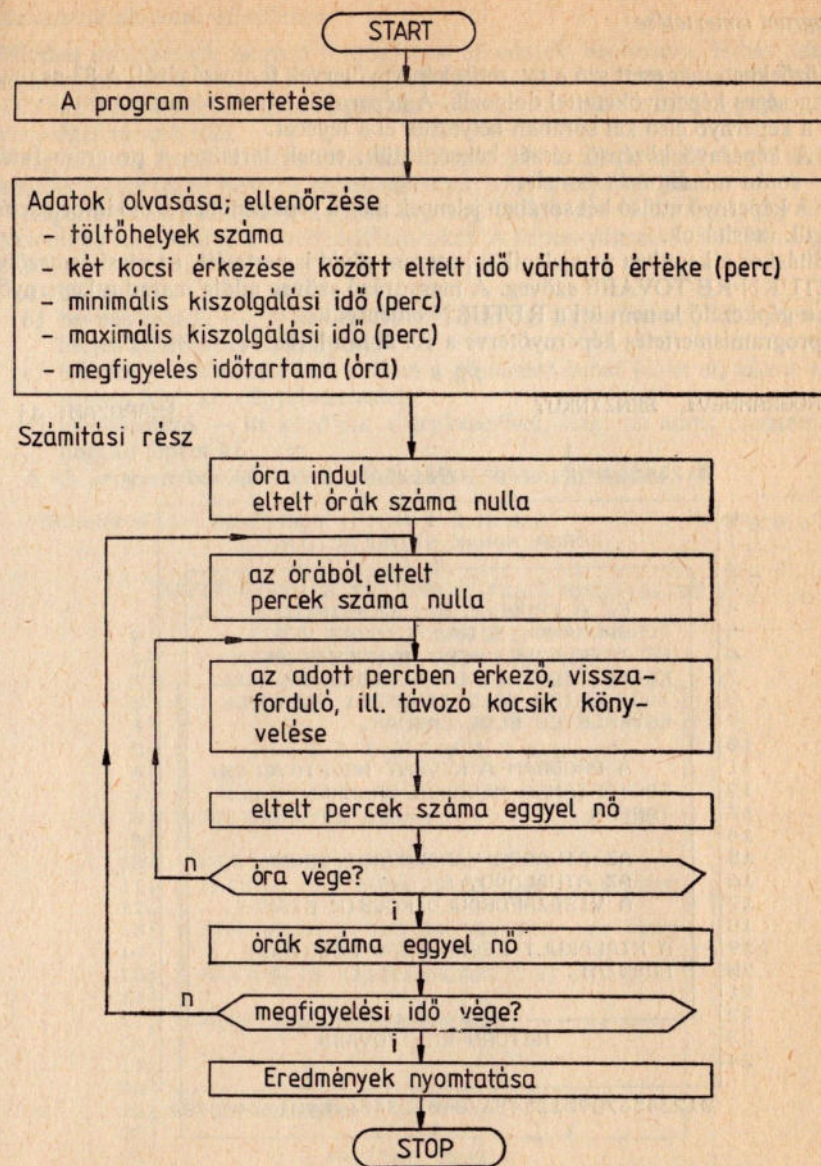
ahol a $[]$ szimbólum az egészrész-függvényt jelenti.

A főprogram (83. program: BENZINKÚT) szerkezetét leíró folyamatábra a 9. ábrán látható.

A program négy elkülönülő részre tagolódik:

- a program ismertetése;
- az adatok olvasása, ellenőrzése;
- számítások;
- az eredmények kiírása.

A továbbiakban külön-külön foglalkozunk a négy rész tervezésével.



9. ábra A főprogram (83. program: BENZINKÚT) szerkezetét leíró folyamatábra

A program ismertetése

Az előzőekben már esett szó a tetszetős képernyőtervek fontosságáról. A 83-as program egységes képernyőkerettel dolgozik. A képernyőt három részre osztottuk:

- a képernyő első két sorában helyeztük el a fejléct.
- A képernyő középső részét bekereteztük, ennek tartalma a program futása során mindig más és más.
- A képernyő utolsó két sorában jelennek meg a gépkezelőnek szóló információk, ill. utasítások.

Indításkor a keretben megjelenik a program rövid ismertetése, az utasításmezőben a RETURN-RE TOVABB szöveg. A magyarázó szöveg addig marad a képernyőn, amíg a gépkezelő le nem üti a RETURN billentyűt.

A programismertetés képernyőterve a 10. ábrán látható.

PROGRAMNEV: BENZINKUT

{LAPSZAM: 1}

	1	2	3
0	123456789012345678901234567890123456789		
1	BENZINKUT		
2	FORGALMANAK SZIMULACIOJA		
3			
4	EZ A PROGRAM EGY BENZINKUT		
5	FORGALMANAK SZIMULACIOJAT VEGZI,		
6	FELTETELEZVE, HOGY A KOCSIK BEER-		
7	KEZESI IDEJET, ES A KISZOLGALASI		
8	IDOT LEIRO VALOSZINUSEGI VALTOZOK		
9	EGYENLETES ELOSZLASUAK.		
10			
11	A PROGRAM A KIVANT MEGFIGYELESI		
12	IDOTARTAMRA MEGADJA ORANKENTI BON-		
13	TASBAN		
14			
15	AZ ATLAGOS VARAKOZASI IDOKET,		
16	AZ ATHALADO, ES		
17	A VISSZAFORDULO KOCSIK SZAMAT,		
18			
19	A KIALAKULT LEGNAGYOBB KOCSISOR		
20	HOSSZAT.		
21			
22	RETURN-RE TOVABB		
23			
24	0123456789012345678901234567890123456789		

MEGJEGYZES: SOTETKEK ALAPON FEHER KERET,
TURKISZ (CYN) BETUK.

10. ábra A programismertetés képernyőmaszkja

Az adatok olvasása, ellenőrzése

Minden programnak nagyon fontos része az adatok beolvasása. Hibás adatokból egyetlen program sem tud helyes eredményeket számítani. A pontos adatbevitel érdekében a gépkezelőnek minden pillanatban pontosan tudnia kell, hogy milyen tevékenységet várunk tőle.

Ugyanilyen fontos a kapott adatok ellenőrzése, a hibák kiszűrése, javítása. Az adatbeolvasás során hasznos segédeszközünk lehet a képernyőmaszk.

Képernyőmaszk alatt egy rögzített képernyőfelépítést értünk, amely egy űrlaphoz hasonlóan tartalmazza a kitöltendő adatokat. A képernyőmaszk kátlalában négy részből áll:

- fejléc;
- beviteli mezők — ide kell beírni az adatokat. Egy adatmezőből a kurzor csak helyes adatbevitel után lép ki;
- hibasor — ha adatbevitel közben a gépkezelő hibát követ el, akkor ebben a sorban írjuk ki a figyelmeztetést;
- utasításmező — itt közöljük a gépkezelővel, hogy az adott programrészből hogyan léphet ki.

A 83. program beviteli képernyőmaszkja a 11. ábrán látható.

PROGRAMNEV: BENZINKUT

{LAPSZAM: 2}

	1	2	3
0	123456789012345678901234567890123456789		
1	BENZINKUT		
2	FORGALMANAK SZIMULACIOJA		
3			
4	ADATBEVITEL		
5			
6			
7			
8	TOLTOHELYEK SZAMA		:X DB
9			
10	KET KOCSI ERKEZESE KOZOTT		
11	ELTELT IDO VARHATO ERTEKE		:XX PERC
12			
13	MINIMALIS KISZOLGALASI IDO		:XX PERC
14			
15	MAXIMALIS KISZOLGALASI IDO		:XX PERC
16			
17	MEGFIGYELES IDOTARTAMA		:XX ORA
18			
19			
20			
21	HIBAJELZES:		
22			
23	JAVITAS/VEGE:X		
24			

0123456789012345678901234567890123456789

MEGJEGYZES: SOTETKEK ALAPON FEHER KERET,
TURKISZ (CYN) SZOVEG, FEHER ADATOK.

11. ábra Adatbeviteli képernyőmaszk

Az adatbevitel folyamata a 83. programban a következő: a képernyőn megjelenik az adatbevitel maszkja. Egy felfelé mutató nyíl jelzi a behívandó adat helyét. A mező kitöltése közben a program ellenőrzi a leütött karaktereket, csak az adat típusának megfelelő karaktereket fogadja el. Az adatmező kitöltése után a kapott adatot ellenőrzi. Amennyiben hibásnak találja, a hiba okát kiírja a képernyőre. Ezután a nyíl az adatmező elejére áll, jelezve, hogy az adatot javítani kell. Helyes kitöltés esetén a nyíl a következő mező elejére ugrik.

Az adatlap kitöltése után a gépkezelő választhat. A V betű leütésével jelzi, hogy az adatbevitel befejeződött, a J betű leütése után pedig lehetőséget kap az adatok javítására. Ebben az esetben az adatbevitelt jelző nyíl újra az első adatmezőre mutat.

A képernyőmaszkkal szervezett beolvasás nagyon kényelmessé teszi a program használatát, de felvet néhány programozástechnikai problémát.

Az INPUT utasítás végrehajtása közben nem tudjuk egyenként ellenőrizni a leütött karaktereket. Ez hibát okozhat pl. akkor, ha numerikus adatmezőbe a gépkezelő nem numerikus karaktert próbál beírni.

Nem ellenőrizhetjük a leütött karakterek számát sem, így nem védekezhetünk a túl hosszú adatok begépelése ellen.

A GET utasítás használatával megoldhatjuk ezeket a problémákat, de mivel a GET utasítás végrehajtásakor nem jelenik meg a kurzor, a beviteli felszólítást valamilyen módon magunknak kell megvalósítanunk. A 83. programban használt beolvasó rutin egy felfelé mutató nyíllal jelzi, hogy hol tart a beolvasás. Mivel a GET utasítás csak egyetlen karaktert olvas be, a beolvasott karaktereket magunknak kell összefűznünk egyetlen változóvá. További problémát jelent, hogy a GET utasítás — az INPUT utasítással ellentétben — nem jelzi ki automatikusan a leütött karaktert. Egnél több karakter bevitele esetén semmiképpen sem kerülhetjük el, hogy a beolvasott karaktert azonnal kiírjuk a képernyőre, hiszen különben a gépkezelő nem látná, hogy helyes adatot gépelt-e be vagy sem.

Amennyiben a GET utasítás mellett döntünk, célszerű készíteni egy alprogramot, amely egy adott hosszúságú, adott típusú változót olvas be.

Az alprogram ellátja az INPUT utasítás funkcióit, elkerülve annak számunkra előnytelen tulajdonságait. A 83-as programban ezt a megoldást választottuk.

A programozás egyik legnehezebb, legkevésbé látványos, ugyanakkor az egyik legfontosabb része az adatok ellenőrzése. Az ellenőrzés első részét a beolvasó alprogramban végezzük el. A

2370 GET A\$

2380 IF (A\$ < "0" OR A\$ > "9") AND A\$ <> " " THEN 2370

utasításokkal kizárjuk, hogy a gépkezelő a számjegyeken és a szóközökön kívül mást is leüthessen.

Az ellenőrzés második részében azt döntjük el, hogy a beolvasott érték az általunk megkívánt határok közé esik-e.

Ellenőrzési szempontjaink a következők:

- a töltőhelyek száma legalább egy és legfeljebb nyolc lehet;
- két kocs érkezése között eltelt idő várható értéke nem lehet egy percnél kevesebb;
- a minimális kiszolgálási idő legalább egy perc;
- a maximális kiszolgálási idő nem lehet kisebb a minimális kiszolgálási idő értékénél;

— a megfigyelés időtartama legalább két óra.

Az adatok ellenőrzése folyamatos, a beolvasással párhuzamosan történik. Minden beolvasott adatot azonnal ellenőrünk. Ha egy adat hibás, akkor meghívunk egy hibajelzést kiíró alprogramot és kérjük az adatbevitel megismétlését.

A lehetséges hibauzenetek szövegét egy füžértömbben helyeztük el. Hiba esetén meghívjuk az alprogramot, és egy változóban átadjuk az aktuális hibauzenet sorszá-
mát.

Az adott percben érkező, visszaforduló, ill. teli tankkal távozó kocsik könyvelésé-
nek folyamatábrája a 12. ábrán látható.

Az eredmények kiírása

Az eredmények kiírása két részből áll. A megfigyelési időszak minden ciklusában ké-
szítünk egy ábrát a töltőállomásokról, és az előttük várakozó kocsikról. Az ábrát a
megszokott keret fogja közre.

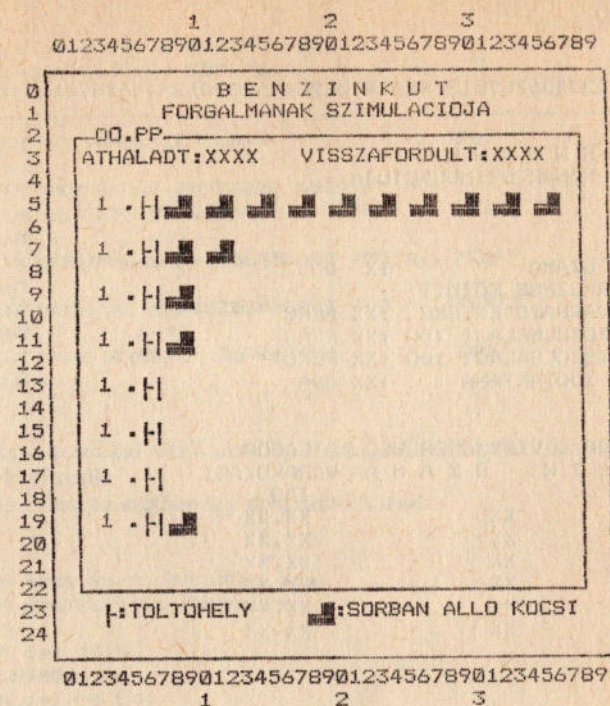
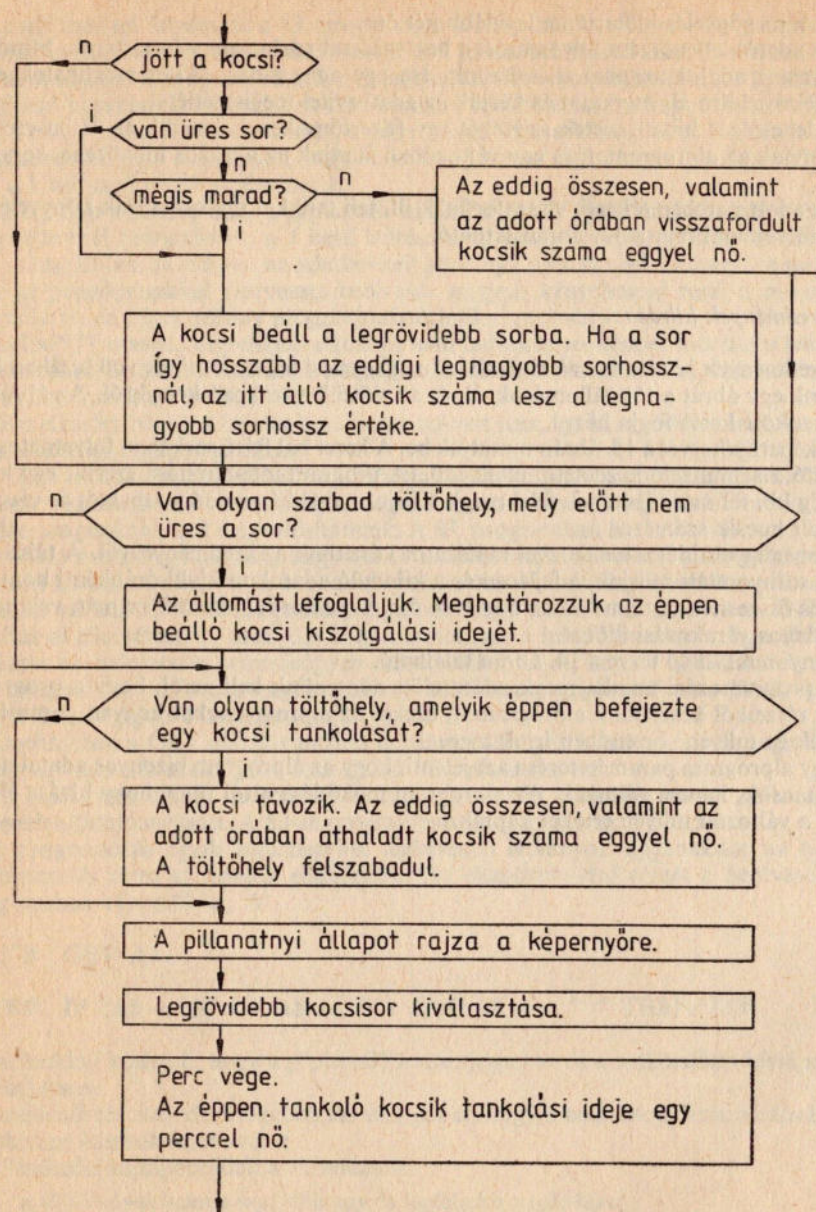
A képernyőtervet a 13. ábrán mutatjuk be. A keret bal felső sarkában folyamatosan
látható a szimuláció kezdetétől eltelt idő. (A program időszámítása szerint egy idő-
egység kb. fél másodpercrek felel meg). Kiírjuk továbbá az eddig áthaladt és vissza-
fordult kocsik számát.

A megfigyelt időszak elteltével táblázatot készítünk az eredményekről. A tábláza-
tot a sornyomtatóra írjuk, a fejlécen és a kiinduló adatokon kívül óránkénti bontás-
ban és összesítve tartalmazza az áthaladó és a visszaforduló kocsik számát, valamint
az átlagos várakozási időket.

A nyomtatáské-
p terve a 14. ábrán található.

A programozási munka megkezdése előtt dönteniünk kell arról, hogy a program
mely részeiből készítsünk alprogramot, ezeket az alprogramokat hogyan paraméte-
rezzük és milyen sorrendben írjuk meg.

Egy alprogram paraméterezése azt jelenti, hogy az alprogram bizonyos adatai nem
konstansok, hanem változók. Az alprogram működése attól függ, hogy hívása előtt
ezek a változók milyen értéket kaptak.



MEGJEGYZES: (OO.PP) AZ ORA

13. ábra Képernyőterv

PROGRAMNEV: BENZINKUT

NYOMTATASTERV

0123456789012345678901234567890123456789012345678901234567890

BENZINKUT
FORGALMANAK SZIMULACIOJA

TOLTOHELYEK SZAMA :X DB
KET KOCSI ERKEZESE KOZOTT
ELTELT IDO VARHATO ERTEKE :XX PERC
MINIMALIS KISZOLGALASI IDO :XX PERC
MAXIMALIS KISZOLGALASI IDO :XX PERC
MEGFIGYELES IDOTARTAMA :XX ORA

ORA	ATHALADO K O C S I K	VISSZAFORDULO S Z A M A	ATLAGOS VARAKOZASI IDO
0.	XX	XX	XX.XX
1.	XX	XX	XX.XX
2.	XX	XX	XX.XX
3.	XX	XX	XX.XX
4.	XX	XX	XX.XX
5.	XX	XX	XX.XX

összesen:

XX	XX	XX.XX
----	----	-------

0123456789012345678901234567890123456789012345678901234567890

14. ábra Nyomtatásterv

Nézzük meg pl. a következő adatbeviteli képmaszkot készítő alprogramot:

```
5600 REM **** 83/1. PROGRAM ****
5602 REM ADATBEVITELI KEPMASZK
5604 :
5610 :
5620 PRINT "TOLTOHELYEK SZAMA" :. DB"
5625 PRINT
5630 PRINT "KET KOCSI ERKEZESE KOZOTT"
5640 PRINT "ELTELT IDO VARHATO ERTEKE :.. PERC"
5645 PRINT
5650 PRINT "MINIMALIS KISZOLGALASI IDO :.. PERC"
5655 PRINT
5660 PRINT "MAXIMALIS KISZOLGALASI IDO :.. PERC"
5665 PRINT
5670 PRINT "MEGFIGYELES IDOTARTAMA :.. ORA"
5680 RETURN
```

Az alprogram sokkal általánosabbá tehető, ha a kiírandó adatnevek és mértékegységek tömbváltozók.

A kiírást ebben az esetben így programozhatjuk:

```
2150 REM **** 83/2. PROGRAM ****
2160 REM ADATBEVITEL KEPMASZK
2165 :
2170 FOR I=1 TO M
2180 PRINTAD$(I);
2190 FOR J=1 TO L(I)
2200 PRINT". ";
2210 NEXT J
2220 PRINT" ";M$(I)
2230 PRINT
2240 NEXT I
2250 RETURN
```

A változók jelentése:

M	Az adatok száma
AD\$(I)	Az i-edik adat neve
M\$(I)	Az i-edik adat mértékegysége
L(I)	Az i-edik adat hossza

A második megoldás első látásra körülményesebbnek tűnik, mégis több szempontból egyszerűsíti a feladat megoldását:

- a szövegkonstansokra egy programon belül általában több helyen szükségünk van. Azzal, hogy ezeket tömbváltozókban helyeztük el, helyet takarítottunk meg;
- ha az adatok szerkezetét módosítanunk kell, nincs szükség a teljes programlista végignézésére, elég az érintett változók értékeit megváltoztatnunk;

c) a megoldás általános, abban az értelemben, hogy hasonló szerkezetű képmaszkok programozására logikai változtatás nélkül, a tömbök tartalmának megfelelő módosításával alkalmazhatjuk.

A paraméterezett alprogramokból kialakíthatunk egy programkönyvtárat. Az alprogramok felhasználásával jelentősen meggyorsíthatjuk az újabb programok készítését.

Ha az alprogramok legfontosabb adatai változók, akkor a módosításokat és illesztéseket könnyedén elvégezhetjük. A kiemelt változók (paraméterek) értékét a konkrét feladatban a program elején rögzítjük. A feladat jellege szerint meghatározott konstansokat elhelyezhetjük értékadó utasításokban vagy DATA sorokban.

Tárolhatjuk a paramétereket lemezen is, ebben az esetben a megváltoztatásukhoz nem kell a programot módosítanunk.

Az alprogramok megírásának sorrendje általában a következő: először elkészítjük (vagy elővesszük a gyűjteményünkéből) az elemi feladatokat végző alprogramokat. Mintapéldánkban ilyen alprogram pl. a kurzor pozicionálása, vagy egyetlen egész típusú változó beolvasása.

Ezután megírjuk a program vázát, majd hozzálátunk az alprogramok kidolgozásához. A kész alprogramokat azonnal ki is próbáljuk, így remélhetjük, hogy a végső fázisban a program már hibátlanul fog működni.

A BENZINKÚT program fontosabb változóinak listája:

N	A töltőhelyek száma
AT	Két kocsni érkezése között eltelt idő várható értéke
K1	Minimális kiszolgálási idő
K2	Maximális kiszolgálási idő
TM	A kút figyelésének időtartama órákban
A(I)	Az <i>i</i> -edik kútnál sorban álló kocsik száma
T(I)	Az <i>i</i> -edik töltőhelynél éppen tankoló kocsni ennyi ideje tankol. Ha $T(I)=0$, akkor az <i>i</i> -edik kút pillanatnyilag szabad.
K(I)	Az <i>i</i> -edik kút kiszolgálási ideje az adott pillanatban
MIN	Annak a kútnak a sorszáma, ahol éppen a legrövidebb sor áll.
T	Az eltelt idő órákban
TT	Az órából eltelt percek száma
MA	Az eddigi legnagyobb sorhossz
V	A visszafordult kocsik száma
A	A kiszolgált kocsik száma
IM	Mutató, $IM=0$, az érkezett kocsni visszafordult, $IM=1$, maradt
VI(T)	A <i>t</i> -edik órában visszaforduló kocsik száma
AT(T)	A <i>t</i> -edik órában áthaladó kocsik száma
M	Az adatok száma
AD\$(I)	Az <i>i</i> -edik adat neve
MS\$(I)	Az <i>i</i> -edik adat mértékegysége
L(I)	Az <i>i</i> -edik adat hossza
H\$(I)	Az <i>i</i> -edik hibaüzenet szövege
H	A hibaüzenet sorszáma
X	A kurzorpozíció sorindexe
Y	A kurzorpozíció oszlopindexe
US	36 szóközből álló füzőr
VS	36 vonalkából (—) álló füzőr
KS	A nyomtatásképi szerkesztéséhez használt füzőr

A BENZINKÚT program szubrutinjainak leírása

I. A képernyőszerkesztés alprogramjai

1. A kurzor pozicionálása (KURZOR)

Az alprogram az (X, Y) koordinátákkal adott képernyőpozícióra állítja a kurzort.

Paraméterek: X, Y.

2. Keret készítése (KERET)

Az alprogram sötétkékre állítja a keret és a háttér színét, kiírja a program fejlécét, és bekeretezi a képernyőt.

Paraméterek: US 36 szóközből álló füzőr;

VS 36 szóközből álló füzőr.

Hívott alprogram: KURZOR.

3. Magyarázat kiírása

Az alprogram a BENZINKÚT program rövid ismertetését írja ki a képernyőre.

Paraméter: nincs.

Hívott alprogramok: KURZOR, KERET.

II. Az adatbevitel alprogramjai

4. Adatnevek, mértékegységek feltöltése

Az alprogram értéket ad az adatszerkezetet leíró változóknak.

Paraméter: nincs.

Hívott alprogram: nincs

Eredmények: M az adatok száma;
AD\$(I) az *i*-edik adat megnevezése;
MS\$(I) az *i*-edik adat mértékegysége;
L(I) az *i*-edik adat hossza.

5. Hibaüzenetek feltöltése

Az alprogram értéket ad a hibaüzeneteket tartalmazó tömb elemeinek.

Paraméter:

Hívott alprogram:

Eredmények: H\$(I) az *i*-edik hibajelzés szövege.

6. Adatbeviteli képmaszk készítése

Az alprogram megjeleníti a képernyőn az adatbevitel képmaszkját.

Paraméterek: M az adatok száma;
AD\$(I) az *i*-edik adat megnevezése;
MS\$(I) az *i*-edik adat mértékegysége;
L(I) az *i*-edik adat hossza.

Hívott alprogramok: KURZOR, KERET

7. Adat olvasása (OLV)

Az alprogram egy adott hosszúságú előjel nélküli egész számot olvas a billentyűzetről. A beolvasandó adat helyét a képmaszkban jelzi, a kapott értéket visszaírja.

Paraméterek: X, Y az adat koordinátái;
L az adat hossza.

Hívott alprogram: KURZOR

Eredmény: F a beolvasott adat.

8. Hibajelzés

Az alprogram háromszor felvillantja a kívánt hibaüzenetet a képernyő 21-edik sorában.

Paraméterek: HS a hibaüzeneteket tartalmazó tömb;
H a kiadni kívánt hibaüzenet sorszáma a HS tömbben;
US 36 szókból álló füžér.

Hívott alprogram: KURZOR

III. Számítási rész

A 9—12. alprogramok közös paramétereket használnak, ezek leírása a változólistában megtalálható. A leírásokban aláhúzással jelöljük azokat a változókat, amelyeknek értékét az ismertett alprogram módosítja.

9. A visszaforduló kocsik könyvelése

Az alprogram eldönti, hogy egy kocsi marad-e vagy visszafordul. Ha a kocsi visszafordul, akkor az eddig összesen és az adott órában visszaforduló kocsik számát megnöveli eggyel. A rutin akkor dönt úgy, hogy a kocsi marad, ha van üres kocsisor, vagy ha a generált véletlen szám értéke kisebb a legrövidebb kocsisor hosszának reciprokánál. Ha pl. a legrövidebb sorban öt kocsi áll, akkor a beérkező kocsi ittmaradásának valószínűsége 20%.

Paraméterek: V, T, VI(T), MIN, A(MIN), IM
VI(T) a VI vektor T-edik elemét jelöli.

10. A tankolás kezdete

Az alprogram megvizsgálja, hogy van-e olyan szabad kút ($T(I)=0$), amely előtt nem üres a kocsisor ($A(I)=0$). Amennyiben az i -edik kút ilyen, lefoglalja ($T(I)=1$) és meghatározza, hogy az éppen megkezdődött kiszolgálás mennyi ideig tart majd. A kiszolgálás időtartama ($K(I)$) a (K_1 , K_2) intervallumba eső véletlen szám.

Paraméterek: N, A[1...N], T[1...N], K₁, K₂, K[1...N]
A[1...N] az A vektor összes elemét jelöli.

11. A tankolás vége

Az alprogram végignézi a töltőhelyeket, és ha olyan kútra talál, ahol a tankolás már befejeződött ($T(I)>K(I)$), akkor a kútat felszabadítja ($T(I)=0$).

A kút előtt álló kocsisorból elenged egy kocsit ($A(I)=A(I)-1$), az eddig összesen, és az adott órában átengedett kocsik számát megnöveli eggyel.

Paraméterek: T, N, A, A[1...N], AT[T], T[1...N], K[1...N]

12. A legrövidebb kocsisor sorszáma meghatározása

Az alprogram megkeresi az A[1...N] vektor legkisebb elemét, és ennek indexét elhelyezi a MIN változóban.

Paraméterek: N, A[1...N], MIN

IV. Eredmények kiírása

13. Rajz

Az alprogram megjeleníti a képernyőn a töltőállomásokat és az előttük várakozó kocsisorokat. Kiírja a megfigyelés kezdetétől eltelt időt, az eddig áthaladt és visszafordult kocsik számát.

Paraméterek: T, TT, A, V, N, A[1...N]

Hívott alprogram: KURZOR

14. Füžér kiegészítése (KIEG)

Az alprogram szóközökkel egészíti ki a K\$ füžért.

Paraméterek: KK a kiegészített füžér;
K\$ a kiegészítendő füžér hossza;
U\$ szóközöket tartalmazó füžér.

A főprogram leírása

I. A programismertetés kiírása, a paramétertömbök feltöltése, az adatok beolvasása

A programrész először kiírja a programismertetést a képernyőre, értéket ad az adatstruktúrát leíró változónak, feltölti a hibaüzenetet tartalmazó tömböt, majd elvégzi az induló adatok beolvasását, javítását.

```
10 REM **** 83. PROGRAM ****
20 REM      BENZINKUT
30 :
40 :
50 V$=" "
60 U$=" "
70 :
80 GOSUB 1870 :REM MAGYARAZO SZOVEG
90 GOSUB 2680 :REM HIBAJELZESEK
100 GOSUB 2750 :REM ADATNEVEK
110 REM **** ADATOK BEOLVASASA ****
120 :
130 GOSUB 2150 :REM ADATBEVITEL KEPMASZK
140 :
150 REM N-----TOLTOHELYEK SZAMA
160 :
170 X=9:Y=31:L=1: GOSUB 2340 :N=F
180 IF N<1 THEN H=1:GOSUB 2530:GOTO 170
190 IF N>8 THEN H=2:GOSUB 2530:GOTO 170
200 :
210 REM AT----KOCSEK ERKEZESE KOZTI IDO
220 :
230 X=12:Y=31:L=2: GOSUB 2340 :AT=F
240 IF AT<1 THEN H=1:GOSUB 2530:GOTO 230
250 :
260 REM K1----MINIMALIS KISZOLGALASI IDO
270 :
280 X=14:Y=31:L=2: GOSUB 2340:K1=F
290 IF K1<1 THEN H=1:GOSUB 2530:GOTO 280
300 :
310 REM K2----MAXIMALIS KISZOLGALASI IDO
320 :
330 X=16:Y=31:L=2: GOSUB 2340:K2=F
340 IF K2<K1 THEN H=3:GOSUB 2530:GOTO 330
350 :
360 REM TM----MEGFIGYELES IDOTARTAMA
370 :
380 X=18:Y=31:L=2: GOSUB 2340:TM=F
390 IF TM<2 THEN H=1:GOSUB 2530:GOTO 380
400 :
410 X=23:Y=25:GOSUB 1640:PRINT"V$ "
420 GET A$:IF A$="" THEN 420
430 IF A$="J" THEN 150
440 :
```


II. Számítási rész

A programrész helyet foglal a számításhoz szükséges tömböknek, és meghatározza az indulóállapothoz tartozó változók értékeit. Kihasználja, hogy a numerikus változók tartalma a program indítása után nulla. Ez éppen azt jelenti, hogy induláskor minden sor üres, a kutak szabadok, a visszafordult és áthaladt kocsik száma nulla. Induláskor az első sort tekintjük a legrövidebbnek ($MIN=1$).

A szimuláció az 530—820-as sorokban folyik, az eltelt órákat a T, az órákból eltelt perceket a TT ciklusváltozó mutatja. A program minden cikluslépésben elvégzi a beérkező, az áthaladó és a visszaforduló kocsik könyvelését, és kirajzolja a kialakult állapotot a képernyőre.

```
450 DIM A(N).T(N).K(N)
460 DIM VI(TM).AT(TM)
470 :
480 GOSUB 1730 : REM KERET
490 X=23:Y=3:GOSUB 1640
500 PRINT" TOLTOHELY SORBAN ALLO KOCSI"
510 MIN=1
520 :
530 FOR T=0 TO TM-1
540 FOR TT=1 TO 60
550 IF RND(0)>1/AT THEN 660
560 :
570 REM KOCSI JOTT.MARAD?
580 GOSUB 1220
590 :
600 IF IM=0 THEN 660
610 :
620 REM A KOCSI BEALL A LEGROVIDEBB SORBA
630 A(MIN)=A(MIN)+1
640 IF A(MIN)>MA THEN MA=A(MIN)
650 :
660 REM TANKOLAS KEZDETE
670 GOSUB 1280
680 :
690 REM TANKOLAS VEGE
700 GOSUB 1350
710 :
720 REM RAJZ
730 GOSUB 1420
740 :
750 REM LEGROVIDEBB KOCSISOR KIVALASZTASA
760 GOSUB 1580
770 :
780 FOR I=1 TO N
790 IF T(I)<>0 THEN T(I)=T(I)+1
800 NEXT I
810 NEXT TT
820 NEXT T
```

III. Eredménytáblázat nyomtatása

A programrész táblázatot készít a számítások végeredményéről. Az eredménylista terve az 5. ábrán látható.

A táblázat óránkénti felbontásban tartalmazza az áthaladó és a visszaforduló kocsik számát, valamint az átlagos várakozási időket.

A nyomtatandó sor szerkesztése a K\$ füzérben először elhelyezzük T értékét. Ezután a KIEG alprogram felhasználásával kiegészítjük szóközzel, hogy a táblázat második oszlopában szereplő AT(T) értéke a nyomtatáskép kilencedik pozíciójára kerüljön. Miután AT(T)-t is hozzáfűztük K\$-hoz, ezt újra szóközzel egészítjük ki, hogy a harmadik oszlopban megjelenő VI(T) is a helyére kerülhessen.

Hasonló módszerrel érjük el, hogy az átlagos várakozási időt ($60/AT(T)$) a táblázat 35. pozíciójától kezdve írja ki programunk.

```
840 REM **** EREDMENYTABLAZAT ****
850 OPEN 4.4
860 PRINT#4," B E N Z I N K U T"
870 PRINT#4," FORGALOM SZIMULACIOJA"
880 PRINT#4
890 PRINT#4:PRINT#4
900 PRINT#4.AD$(1);N;M$(1)
910 PRINT#4.AD$(2);AT;M$(2)
920 PRINT#4.AD$(3);K1;M$(3)
930 PRINT#4.AD$(4);K2;M$(4)
940 PRINT#4.AD$(5);TM;M$(5)
950 PRINT#4:PRINT#4
960 K$=""
970 K$="ORA ATHALADO VISSZAFORDULO ATLAGOS"
980 PRINT#4.K$
990 K$=" K O C S I K S Z A M A VARAKOZASI"
1000 PRINT#4.K$
1010 PRINT#4.LEFT$(U$,33);"IDO"
1020 FOR T=0 TO TM-1
1030 K$=STR$(T)+".":KK=8:GOSUB 2920
1040 K$=K$+STR$(AT(T)):KK=19:GOSUB 2920
1050 K$=K$+STR$(VI(T)):KK=34:GOSUB 2920
1060 S=INT(60/AT(T)*100+.5)/100
1070 K$=K$+STR$(S)
1080 PRINT#4.K$
1090 NEXT T
1100 PRINT#4:PRINT#4,"OSSZESEN:"
1110 K$=LEFT$(U$,8)
1120 K$=K$+STR$(A):KK=19:GOSUB 2920
1130 K$=K$+STR$(V):KK=34:GOSUB 2920
1140 S=INT((60*TM)/A*100+.5)/100
1150 K$=K$+STR$(S)
1160 PRINT#4.K$
1170 PRINT#4
1180 PRINT#4," A LEGHOSSZABB SOR ";MA;
1190 PRINT#4," KOCSIBOL ALLT"
1200 CLOSE 4
1210 END
```


A 83. program alprogramjainak listája:

```

1220 IM=1
1230 IF A(MIN)=0 THEN 1260
1240 IF RND(0)<1/A(MIN) THEN 1260
1250 V=V+1:VI(T)=VI(T)+1:IM=0
1260 RETURN
1270 :
1280 FOR I=1 TO N
1290 IF A(I)=0 OR T(I)<>0 THEN 1320
1300 T(I)=1
1310 K(I)=INT(RND(0)*(K2-K1)+K1)+1
1320 NEXT I
1330 RETURN
1340 :
1350 FOR I=1 TO N
1360 IF T(I)<=K(I) THEN 1390
1370 A(I)=A(I)-1:T(I)=0
1380 AT(T)=AT(T)+1:A=A+1
1390 NEXT I
1400 RETURN
1410 :
1420 X=3:Y=3:GOSUB 1640
1430 PRINT "X";T;"Y";TT;"X"
1440 PRINT "ATHALADT:";A
1450 X=4:Y=18:GOSUB 1640
1460 PRINT "VISSZAFORDULT:";V
1470 FOR I=1 TO N
1480 X=(I-1)*2+6:Y=2:GOSUB 1640
1490 PRINT I;"X";Y
1500 IF A(I)=0 THEN 1540
1510 FOR J=1 TO A(I)
1520 PRINT "X";Y
1530 NEXT J
1540 PRINT " "
1550 NEXT I
1560 RETURN
1570 :
1580 REM **** LEGROVIDEBB KOCSISOR ****
1590 :
1600 FOR I=1 TO N
1610 IF A(I)<A(MIN) THEN MIN=I
1620 NEXT I
1630 RETURN
1640 REM KURZOR POZICIONALAS
1650 REM X-----SOR (0-24)
1660 REM Y-----OSZLOP (0-39)
1670 :
1680 POKE 214,X:POKE 211,Y
1690 SYS 58640
1700 RETURN
1710 :

```

```

1730 REM KERET
1740 :
1750 POKE 53280,6:POKE 53281,6
1760 PRINT "BENZINKUT"
1770 PRINT "BENZINKUT"
1780 PRINT "FORGALOM SZIMULACIOJA"
1790 PRINT "V:"
1800 FOR J=1 TO 18
1810 PRINT "U:"
1820 NEXT J
1830 PRINT "V:"
1840 RETURN
1850 :
1860 :
1870 REM MAGYARAZO SZOVEG
1880 :
1890 GOSUB 1730: REM KERET
1900 X=5:Y=5:GOSUB 1640 : REM KURZOR POZ.
1910 PRINT "TURKISZ(CYN)"
1920 PRINT "EZ A PROGRAM EGY BENZINKUT"
1930 PRINT "FORGALMANAK SZIMULACIOJAT VEZIR"
1940 PRINT "FELTETCILEZVE, HOGY A KOCSIK BEER-"
1950 PRINT "KEZESI IDEJET,ES A KISZOLGALASI"
1960 PRINT "IDOT VALOSZINUSEGI VALTOZOK"
1970 PRINT "EGYENLETES ELOSZLASUAK"
1980 PRINT
1990 PRINT "PROGRAM A KIVANT MEGFIGYELESI"
2000 PRINT "IDOTARTAMRA MEGADJA ORANKENTI BON-"
2010 PRINT "TASBAN"
2020 PRINT
2030 PRINT "AZ ATLAGOS VARAKOZASI IDOKET,"
2040 PRINT "AZ ATHALADO ES"
2050 PRINT "VISSZAFORDULO KOCSIK SZAMAT,"
2060 PRINT
2070 PRINT "KIALAKULT LEGNAGYOBB KOCSISOR"
2080 PRINT "HOSSZAT"
2090 X=23:Y=12:GOSUB 1640
2100 PRINT "RETURN-RE TOVABB"
2110 GET A$:IF A$<>CHR$(13) THEN 2110
2120 RETURN
2130 :
2140 :
2150 REM ADATBEVITEL KEPASZK
2160 :
2170 GOSUB 1730: REM KERET
2180 X=5:Y=9: GOSUB 1640
2190 PRINT "ADATBEVITEL"
2200 X=9 :Y=0:GOSUB 1640
2210 FOR I=1 TO M
2220 PRINT "AD:"
2230 FOR J=1 TO L(I)
2240 PRINT " "
2250 NEXT J
2260 PRINT "M:"
2270 PRINT
2280 NEXT I
2290 X=23:Y=12:GOSUB 1640
2300 PRINT "AVITAS/VEGE:"
2310 RETURN

```


A 83. program futtatási eredménye nyomtatón:

BENZINKUT
FORGALOM SZIMULACIOJA

TOLTOHELYEK SZAMA : 6 DB
KET KOCSI ERKEZESE KOZOTT
ELTELT IDO VARHATO ERTEKE : 1 PERC
MINIMALIS KISZOLGALASI IDO : 5 PERC
MAXIMALIS KISZOLGALASI IDO : 20 PERC
MEGFIGYELES IDOTARTAMA : 8 ORA

ORA	ATHALADO K O C S I K	VISSZAFORDULO S Z A M A	ATLAGOS VARAKOZASI IDO
0.	21	20	2.86
1.	26	36	2.31
2.	25	32	2.4
3.	25	35	2.4
4.	26	35	2.31
5.	29	31	2.07
6.	26	33	2.31
7.	28	30	2.14
OSSZESEN:	206	252	2.33

A LEGHOSSZABB SOR 4 KOCSIBOL ALLT

4. FEJEZET

Adatfeldolgozás

A statisztika azt mutatja, hogy a világon a gépkapacitások kb. 70%-át adatfeldolgozási feladatok kötik le. Ez a tény már önmagában is indokolja, hogy ezzel a területtel részletesebben foglalkozzunk.

Mit értünk számítógépes adatfeldolgozás alatt? A számítógép, háttértárolók segítségével nagy tömegű adat tárolására alkalmas. Azonban ez az adattömeg „ömlesztve”, szűrés, válogatás, csoportosítás nélkül, szinte hasznavehetetlen lenne. Az adatfeldolgozás a számítógép felhasználásának olyan módja, amely lehetővé teszi, hogy a tárolt adatok a kívánt területeken használható információk formájában jelenjenek meg. Magában foglalja az informatika, a rendszertervezés, a programtervezés és az üzemeltetés szervezésének jelentős részeit, természetesen az alapvető számítástechnikai hardver, szoftver ismereteken kívül. Az adatbankok lehetőséget nyújtanak egy kutatónak arra, hogy pl. számítógépén lekérje mindazon szakcikkek jegyzékét, amelyek az ő témájával foglalkoznak. Így elkerülhető, hogy egy már megjelent és publikált kutatási eredményt, találmányt újból felfedezzünk.

A tsz-elnökök ugyancsak eredményesen használhatják saját adatbankjukat, ha napra készen szeretnék tudni a raktárban levő alkatrészek számát, ami aratás idején különösen fontos. Nagyon sok munkát és fáradtságot, sőt pénzt is meg lehetne takarítani, ha az anyagbeszerzők ezen anyagokat nem saját maguk szereznék be (ha ugyan tudják, hogy hová kellene menni érte), hanem egy terminálon keresztül lekérdezhetnék, hogy az országban hol kapható a keresett alkatrész és milyen határidővel tudják szállítani. A legkedvezőbb feltételt kínáló alkatrészszállítónak feladhatják terminálon a rendelést, s a rendszer az elszámolást is automatikusan elintézné.

Országos számítógép-hálózatra kapcsolt adatbankoknak a magánemberek is sok hasznát vehetnék, pl. tanácsi és OTP szolgáltatások, közművelődési információk stb. az ország bármely részén azonos módon juthatnának el a felhasználókhoz. Az adatfeldolgozás alkalmazhatóságának feltétele, s egyben a rendszer jósága nagymértékben függ a szolgáltatott adatok valóságától, s a rendszer naprakész állapotától.

Közömbös, hogy milyen programnyelven dolgozunk (BASIC, FORTRAN, PL/1, PASCAL stb.), az adatkezelés, az adatfeldolgozás minden nyelven egyaránt fontos szerepet játszik. Ebben a fejezetben áttekintjük a COMMODORE 64-es adatkezelési sajátosságait. Az elméleti leírásokat mindvégig gyakorlati feladatokkal illusztráljuk, bízva abban, hogy ez elősegíti a könnyebb megértést. Először ismerkedjünk meg az adatfeldolgozás fontosabb alapfogalmaival!

a) A file

Manapság az adatfeldolgozási szakkifejezések (file, rekord, mező stb.) szinte hétköznapi fogalmakká váltak. Mit takar a file fogalom? Magyarázatként nézzünk egy példát!

Az egyetemi hallgatók nyilvántartásához egy olyan adatállományra van szükségünk,

amelynek tartalma pl. megegyezik a hallgatók személyi igazolványának tartalmával. Ezt elképzelhetjük úgy is, hogy az igazolványokat egy nagy dobozba rakjuk, s esetleg ábécé sorrendbe rendezzük. A személyi igazolványok halmaza a dobozzal együtt analóg a file fogalmával. Tehát a file egy nyilvántartásban lévő adathalmaz számítás-technikai megfelelője. (A szó angol eredetű, magyar megfelelője az adatállomány, adathalmaz).

b) A rekord

Az előző példánál maradva, egy rekord megfelel egy személyi igazolványnak. Az egyes rekordok szerkezete (mint minden személyi igazolványé) megegyezik. (Természetesen a tartalmuk különböző).

Ezek alapján mondhatjuk, hogy egy file rekordokból áll!

c) A mező

Azt tudjuk, hogy az egy file-ba tartozó rekordok szerkezete azonos. Mit jelent ez pontosabban? Azt, hogy mindegyik rekordban (személyi igazolványban) található egy név, egy születési év, egy anyja neve stb. „rovat”, ahová a tulajdonos megfelelő adatai kerülnek. Egy-egy ilyen rovatot mezőnek nevezzük. A mezőkben tárolt információ állhat csak betűkből (pl. a név mező), csak számokból (pl. a születési év mező) vagy mindkettőből (pl. a személyi igazolvány száma). Ezek alapján három mezőtípust különböztetünk meg:

- csak betűkből áll a karakteres mező;
- csak számokból áll a numerikus mező;
- vegyes az alfanumerikus mező.

Tehát egy rekord különböző típusú mezőkből állhat. Azt, hogy milyen mezőket tartalmazzon egy rekord, mindig a konkrét feladat célja határozza meg. Az elmondottak alapján a file-t összetartozó, azonos típusú adatmezőkből álló rekordok halmazaként is definiálhatjuk.

4.1 A LEMEZ FELÉPÍTÉSE

Az új lemezeket, mint már említettük, nem lehet előkészítés nélkül programok, ill. adatok tárolására használni. Az előkészítési folyamatot *formázásnak* nevezzük. Mit takar ez a fogalom? A különböző típusú lemezegységek más-más mechanikus elven működnek. Általában különböző típusú egységeken használt lemezek nem cserélhetők fel, azaz az egyik típusú egység nem képes a másik által „felírt” információk „elolvasására”. (Tehát pl. egy IBM PC-n felírt program C 64-es gépen értelemeszerűen nem olvasható be). Ennek egyik oka az, hogy a lemezek információhordozó területeinek felosztása a különböző típusú egységeken eltérő. Minden lemez koncentrikus kör alakú *sávokra* és azon belül *szektorokra* van felosztva. A sávok és a szektorok száma az adott típusú gépre jellemző adat. A formázás során minden szektor egy ún. *címet* kap, amellyel a lemezegység operációs rendszere (továbbiakban DOS) az adott lemezterületet azonosítja. A cím az adott sáv, ill. szektor sorszámból tevődik össze. Az egyes szektorokba azonosító információként egy 2 karakterből álló kód is felíródik, ez az ID (l. a NEW parancs leírásánál). A lemez 18-as sávjának 0-as szektorában található még egy jellemző azonosító kód, amelynek alapján a DOS felismeri, hogy a lemez formálása a VC—1541-es egységen történt. A VC—1541-es lemezegységnél ez a kód mindig 2A. A szektor további területén 256 karakterből álló in-

formáció tárolható. A formálás utolsó lépése a tartalomjegyzék (directory) kialakítása, amely a lemez 18-as sávján foglal helyet.

A VC—1541-es lemezegység néhány paramétere:

a sávok száma	35
a byte-ok száma szektoronként	256
a szektorok (blokkok) száma	683
a szabad szektorok száma	664 (a többi a tartalomjegyzék foglalja le.)
a file-ok max. száma lemezenként	144

A szektorok száma függ a sávok számától. Az 1-es sáv a legkülső sáv, s egyben a legnagyobb, a 35-ös sáv a legbelső, s egyben a legkisebb. Az egyes sávokon található szektorok számáról a következő táblázat ad bővebb felvilágosítást.

A sávok száma	A szektorok száma
1—17	21
18—24	19
25—30	18
31—35	17

A VC—1541-es egységnek a nagyobb kategóriájú lemezegységekhez hasonlóan saját processzora és operációs rendszere van. Az operációs rendszer, — a DOS — nem foglal el külön tárterületet az alapgép központi tárából. További érdekessége és ugyanakkor előnyös tulajdonsága, hogy a lemezműveleteket teljesen önállóan, az alapgép igénybevétele nélkül hajtja végre. Minthogy azonban így a lemezműveletekre vonatkozó utasítások nem szerepelnek a C 64-es utasításkészletében, ezeket az utasításokat valamilyen módon el kell juttatni a lemezegységhez. Átvitel után ezek az utasítások meghívják a kívánt feladatot ténylegesen elvégző DOS rutinokat.

Minden utasítást vagy parancsot egy ún. *csatornán* keresztül küldünk el a lemezegységhez. A rendelkezésre álló 16 csatorna közül a 15-ös számú a parancscsatorna (vagy utasításcsatorna).

Az adatvitel lépései:

— a csatorna megnyitása	(OPEN)
— adatátvitel	(PRINT)
— a csatorna lezárása	(CLOSE)

Az OPEN utasításban mindig meg kell adni annak az egységnek a fizikai számát, amelyhez a parancsot küldjük, továbbá egy ún. *logikai file-számot*, amelyhez a parancscsatornát hozzárendeltük.

Az

OPEN logikai file szám, 8,15, „utasítás”

parancs alapján a címzett egység a 8-as, a parancscsatorna a 15-ös, a logikai file-szám értéke 1—127 közé eső tetszőleges egész szám. Ezt az átviteli utasításokban (PRINT#, INPUT#, GET#) használjuk.

4.1.1 A lemezegység rendszerutasításai

NEW — lemez formálása. Rövidítve: **N**:

Az utasítás formátuma:

NEW: lemeznév, azonosító

Az azonosító helyett gyakran használjuk az angol identifier (azonosító) szóból eredő ID rövidítést.

A lemeznév: max. 16 karakterből állhat és bekerül a lemez tartalomjegyzékének fejlécébe.

Az azonosító (ID): két tetszőleges karakter, amelynek alapján a DOS felismeri, ha lemezegységben munka közben lemezt cseréltünk. Pl.:

OPEN1,8,15,"N:TESZTLEMEZ,TL"

A RETURN billentyű megnyomása után az egység egy sajátos berregő hangot ad, majd kb. 80 másodpercig „dolgozik” a lemezen. Ezalatt a lemezegységen levő piros LED világít, jelezvén, hogy az író-olvasó fej a lemezen van.

SCRATCH — file-ok törlése. Rövidítve: **S**:

Azokat a file-okat, amelyekre már nincs szükségünk, törölhetjük a lemezeről.

Az utasítás formátuma:

S: file-név 1, file-név 2,...

Pl.: az

OPEN1,8,15,"S:TESZT"
CLOSE 1

a "TESZT" nevű file-t törli a lemezeről.

RENAME — file-ok névmódosítása. Rövidítve: **R**:

A paranccsal megváltoztathatjuk egy file nevét a tartalomjegyzékben.

Az utasítás formátuma:

R: új név=régi név

Pl.: az

OPEN1,8,15,"R:TESZT01=TESZT"
CLOSE1

parancsok végrehajtása után a "TESZT" nevű file új neve: "TESZT01"

COPY — file-ok másolása. Rövidítve: **C**:

Ezzel az utasítással egy file egy lemezen belül másolható, duplikálható, természetesen más néven, ill. több kis file egy file-ba összefűzhető.

Az utasítás formátuma:

C: új file=régi file1, régi file2...

Pl.:

OPEN1,8,15,"C:KIADÁS=KIAD1,KIAD2,KIAD3"
CLOSE1

Ezt az összefűzési módszert programoknál nem alkalmazhatjuk!

INITIALIZE — a lemez inicializálása. Rövidítve: **I**:

A DOS a lemez minden szektoráról nyilvántartja, hogy az foglalt-e vagy sem. Ez a nyilvántartás a lemez 18-as sávjának 0. szektorában, az ún. BAM-ban található (BAM=Block Availability Map). A BAM-ot a lemezegység a lemez behelyezésekor a RAM területre betölti. A lemezcserét a DOS az ID alapján veszi észre, s ezt a töltést automatikusan elvégzi. Ha azonban a lemez ID azonos, akkor ezt nem teszi meg, ekkor mi tölthetjük be a BAM-ot az INITIALIZE paranccsal.

Az utasítás formátuma:

I:

Pl.:

OPEN1,8,15,"I:"
CLOSE1

VALIDATE — a lemez újraszervezése. Rövidítve: **V**:

Ha a lemezen már sokszor töröltünk file-okat, vagy írtunk adatokat a lemeze, de nem zártuk le a file-t, akkor előfordulhat, hogy a BAM-ban néhány szektor foglalt-nak minősül, pedig a tartalomjegyzékben egyetlen file-hoz sem tartozik. Jó lenne ezeket a szektorokat felszabadítani, hiszen egyébként a lemezen a szabad terület állandóan csökkenne. Ezt végzi a VALIDATE parancs hatására a DOS. Első lépésben minden szektort felszabadít, majd végigmegy a tartalomjegyzékben szereplő összes file-on, megkeresi a hozzájuk tartozó szektorokat, s azokat újra lefoglalja a BAM-ban.

Az utasítás formátuma:

V:

Pl.:

OPEN1,8,15,"V:"
CLOSE1

Ezek után a lemezegység „dolgozik”, mégpedig a tartalomjegyzékben szereplő file-ok számától és méretétől függő ideig.

4.2 SOROS (SZEKVENCIÁLIS) FILE SZERVEZÉSE

Ez a legegyszerűbb file-kezelési eljárás. A neve az adatkezelés módjára, a soros (szekvenciális) adatelérésre utal.

Vizsgáljuk meg ezt az adattárolási formát egy analóg példán keresztül! Ha egy magnókazettán levő zeneszámok halmazát egy file-nak tekintjük, akkor az egyes rekordoknak a szalagon levő zeneszámokat feleltethetjük meg. Az egyes zeneszámokat sorban meghallgathatjuk a szalag elejétől a végéig. A probléma akkor jelentkezik, ha csak egy meghatározott zeneszámmra (rekordra) vagyunk kíváncsiak. Ezt csak úgy tudjuk megvalósítani, hogy a szalag elejétől áttekerjük a zeneszámokat, vagyis a szalagot a kívánt helyre pozicionáljuk. Természetesen ez elég lassú eljárás, s ez a soros adattárolás egyik hátránya. Vagyis (most már az előbbi példától függetlenül), ha az adatokat egy soros file-ban tároljuk, akkor egy konkrét rekordot csak úgy tudunk elérni, ha az előtte levő (számunkra most lényegtelen) rekordokat átolvassuk.

Az eddigiekből már látható, hogy soros adatszervezést csak akkor célszerű alkalmazni, ha az adatelérés módja egyféleképpen rögzített. Ilyen esetben ez az adattárolási forma kitűnően alkalmazható. Van azonban a soros file szervezésénél egy lényeges gond, amely az adathalmaz javításakor áll elő. Képzeljük el, hogy a soros adatállomány egy rekordját szeretnénk javítani (pl. azért, mert az ott tárolt adatok időközben megváltoztak, aktualitásukat veszítették). Ekkor a kívánt rekordra pozicionálunk és az új rekorddal a régit felülírjuk. Tehát úgy járhatunk el, mint amikor pl. egy magnókazetta közepéről egy zeneszámot szeretnénk törölni, s a helyére egy újat veszünk fel. Vigyázat! Ha az új zeneszám hosszabb, mint a régi (tehát több helyre van szükség a kazettán), akkor könnyen letörölhetjük a következő zenedarab egy részét. Ha rövidebb, akkor pedig az előző régiből marad utána egy zenerészlet, ami rendkívül zavaró.

Hasonló a helyzet a soros file módosítása során is, ha az új rekord mérete nem egyezik a régi méretével. De mi történik, ha az új és a régi rekord mérete garantáltan megegyezik? Nos, ekkor sem tudjuk az új rekorddal a régit pontosan felülírni, legalább is erre 100%-os garanciát nem vállalhatunk. Miért? A választ a háttértároló mechanikai szerkezeti elemeinek pontatlanságában, ill. a pozicionálást követő megállás utáni adathordozó csúszásban kell keresni. Vagyis „hajszálpontosan” nem tudunk a kívánt rekordra pozicionálni. Ezt a számítógép úgy védi ki, hogy az egyes rekordok között üres helyeket hagy.

Itt értékes információt nem tárolunk, s az üres hely hossza is változó. A rekord kezdetét és végét a számítógép speciális jelek alapján felismeri, s az adatbevitel kezdetét és végét ezek révén vezérli.

Ezek szerint a soros file-t nem lehet módosítani? Erről szó sincs! A módosítást, javítást teljes biztonsággal végrehajthatjuk.

A könnyebb érthetőség kedvéért nézzünk erre egy példát! Tekintsük az ADAT1 nevű soros file-t, amely 500 rekordot tartalmaz. Feladat: a 237-edik rekord felülírása egy új rekorddal. Az új rekordot billentyűzetről visszük be.

Megoldás: egy ADAT2 nevű állományba átmásoljuk az ADAT1 nevű állományt a 236. rekordjával bezárólag. Billentyűzetről beolvassuk a 237. rekordot, s kiírjuk az ADAT2 nevű állományba. Beolvassuk az ADAT1-ből a 237. (hibás) rekordot, s nem írjuk ki sehová. Ezek után folytathatjuk a másolást az ADAT1 238-adik rekordjától az 500-adik rekordig, s ezeket rendre kiírjuk az ADAT2 állományba.

A bemutatott módszerrel tetszőleges javításokat el tudunk végezni a soros file-ban.

4.2.1 A soros file megnyitása

A file-műveletek elvégzése előtt a file-t meg kell nyitni. A megnyitást az OPEN parancssal, ill. utasítással végezhetjük el. A formátuma a következő:

OPEN *lfn,8,mc,"file-név,típus,mód"*

<i>lfn</i>	a logikai file-szám. 1—127 közötti tetszőleges szám, ezzel a számmal hivatkozhatunk a BASIC programban az adott file-ra;
8	a lemezegység fizikai száma. Ha több lemezegységet kötöttünk össze, akkor értelemszerűen annak az egységnek a számát kell beírni, amelyikben a kívánt adatállomány van;
<i>mc</i>	másodlagos cím. Ezzel jelöljük ki a lemezegységnek azt a csatornát, amelyen keresztül az adatokat továbbítja. A 0-s és 1-es másodlagos címet az operációs rendszer a programok tárolásához és betöltéséhez tartja fenn. A 15-ös az utasítás és hibacsatornára vonatkozik;
<i>file-név</i>	max. 16 karakter hosszú név. Ezen a néven kerül be a file a lemez tartalomjegyzékébe. Ha ilyen nevű file már létezik a lemezen, s azt felül akarjuk írni, akkor hasonlóan a SAVE utasításhoz, a @ : jelet a file-név elé kell írni;
<i>mód</i>	ez a paraméter határozza meg a megnyitott csatorna használatának módját. A használható jelölések: W egy soros file-t írásra nyitunk meg, R egy soros file-t olvasásra nyitunk meg, A egy soros file-t bővítésre nyitunk meg, M egy még nem lezárt soros file olvasása.

Egy lemezegységen egyszerre három soros file-t nyithatunk meg. Azonban logikai file-szám (*lfn*) mellett a másodlagos címeknek (*mc*) is feltétlenül eltérőeknek kell lenniük az egyes OPEN utasításokban.

Példák a file megnyitására

a) OPEN1,8,2,"ADAT1,S,W"

A parancs végrehajtása után megnyitottunk a lemezen egy ADAT1 nevű soros file-t írásra. Erről könnyen meggyőződhetünk, ha betöltjük a lemez tartalomjegyzékét.

LOAD"\$",8
LIST

A tartalomjegyzékben a következőket fogjuk látni:

0 ADAT1 *SEQ

ahol:

SEQ a file típusára utal (szekvenciális);
ADAT1 ilyen néven hoztuk létre az adathalmazt;
0 az adathalmaz által elfoglalt blokkok száma;

- * a file-t megnyitottuk, de még nem zártuk le. Tehát hiába írtunk volna ki rekordokat az ADAT1 nevű soros file-ba, ha nem zárjuk le, akkor az használhatatlan lesz.

b) OPEN1,8,2,"ADAT1,S,R"

Egy soros file megnyitása olvasásra.

c) OPEN1,8,2,"ADAT1,S,A"

Egy soros file megnyitása bővítésre. Ezt akkor használjuk, ha a file már létezik, csak további rekordokkal szeretnénk bővíteni a régiek megtartása mellett. Ilyenkor az új rekordokat automatikusan az adathalmaz végéhez illeszti a rendszer. Egy soros file-t tetszőleges sokszor megnyithatunk bővítésére.

d) OPEN1,8,2,"ADAT1"

Az ADAT1 nevű soros file megnyitása olvasásra. (Tehát adott file-típus esetén a mód alapértelmezése az olvasás —, R.)

4.2.2 A soros file lezárása

Ha egy file-ba már nem írtunk többet, vagy már nem olvasunk belőle több rekordot, nem használjuk a file-t a programban tovább, akkor azt le kell zárni. A file lezárását a CLOSE BASIC utasítással végezzük el. Ennek formátuma a következő:

CLOSE *lfn*

ahol:

lfn az OPEN utasításban megadott logikai file-szám.

Ha a CLOSE utasítást nem adjuk ki, s programunk enélkül ér véget, akkor a következőkkel számolhatunk:

a) ha a file olvasásra volt megnyitva (mód=R), akkor különösebb probléma nem lép fel. Azért észrevehetjük ezt a hibát úgy, hogy a lemezegységen levő piros LED nem alszik el. A programban ez különösebb hibát nem is okoz. Viszont, ha egy másik programrész ugyanazt a logikai file-számot használja az OPEN utasításában, akkor a számítógép a

FILE OPEN ERROR

hibaüzenettel figyelmeztet arra, hogy az OPEN utasítás végrehajtása során hibát érzékelt;

b) ha a file írásra volt megnyitva (mód=W), akkor ez már súlyosabb hiba.

Az előző pont első példájában láthattuk, hogy egy megnyitott file-típus jelzése elé egy * karakter került. Az ilyen írásra megnyitott, de le nem zárt file-t nem tudjuk újból megnyitni, ill. az abban tárolt rekordokat nem tudjuk beolvasni. Nézzünk erre is egy példát! Az

OPEN1,8,2,"XX,S,W"

parancs végrehajtása után a lemezen megnyitunk egy XX nevű soros file-t. Majd írjuk be a következő programot:

```
10 OPEN1,8,2,"XX,S,R"
20 I=I+1
30 PRINT I
```

Futtassuk a programot! Az OPEN utasításon a számítógép túlment, hiszen a képernyőre kiírja az I változó értékét (1), azonban a lemezen levő piros LED szaporán villogni kezd. A hiba okát a hibacsatorna leolvasásával tudhatjuk meg. A hiba-üzenet:

WRITE FILE OPEN

ami annyit jelent, hogy a kérdéses file még írásra nyitva van. Ha a LED a hibacsatorna leolvasása után folyamatosan égve maradt, akkor a CLOSE *lfn* paranccsal lezárhatjuk a file-t. Ha erre már nincs módunk, úgy az OPEN utasítást a mód=M paraméterrel alkalmazzuk, s a hibás (*-os) file-t egy másik file-ba átmásoljuk, szabályosan lezárjuk. Így adatvesztés nélkül megmenthetjük a „csillagos” file-t. De ha a file tartalma nem lényeges, le is törölhetjük a lemezzel a SCRATCH paranccsal. Ha viszont sok csillagos file van a lemezen és egyik tartalma sem lényeges, akkor az egyenként való törlések helyett alkalmazhatjuk a VALIDATE lemezparancsot, amely egyszerre eltünteti az összes csillagos file-t.

4.2.3 Írás soros file-ba

Egy írásra megnyitott file-ba a PRINT utasítással írhatunk adatokat. A PRINT utasítás végrehajtásával a DOS egy RETURN karaktert is továbbít, amely az adatok elválasztásához szükséges. Az utasítás formátuma:

PRINT#*lfn*, *adat*

ahol:

#*lfn* az OPEN utasításban megadott logikai file-szám.

Példák

a) Írjunk programot, amely az ADAT1 nevű soros file-ba beírja 1-től 100-ig az egész számokat!


```

5 REM 84. PROGRAM
10 OPEN1,8,2,"@:ADAT1,S,W"
20 FOR I=1 TO 100
30 PRINT#1,I
40 NEXT I
50 CLOSE1

```

A 10-es sorban megnyitottuk az ADAT1 nevű soros file-t írásra. Ha már létezik ilyen nevű file a lemezen, úgy azt felülírjuk.

A 30-as sorban a ciklusváltozót kiírjuk az lfn=1 számú file-ba, majd a ciklus lefutása után lezárjuk a file-t.

Futtassuk le a programot, és nézzük meg az eredményt a lemez tartalomjegyzékében!

b) Írjunk programot, amely a NEVSOR nevű soros file-ba felviszi egy tanulókor hallgatóinak nevét és születési évszámát! Az adatokat billentyűkről vesszük be.

```

5 REM 85. PROGRAM
10 OPEN1,8,2,"NEVSOR,S,W"
20 INPUT"NEV          : ";N$
30 INPUT"SZULETESI EV: ";EV
40 IF N$="" AND EV=0 THEN 200
50 PRINT#1,N$
60 PRINT#1,EV
70 N$="":EV=0
80 GOTO 20
200 REM ADATBEVITEL VEGE
210 CLOSE1
220 STOP

```

- 10 A NEVSOR nevű file-t megnyitjuk írásra.
- 20—30 Adatok bevitel billentyűkről.
- 40 Az adatbevitel végét az üres mezőkkel jelöltük. Ennek lekérdezése.
- 50—60 A beolvasott adatok kiírása lemeze.
- 70 Az input változók törlése. Erre azért van szükség, mert az INPUT utasítás a változók korábbi értékeit megtartja, ha arra csak a RETURN billentyűvel válaszolunk.
- 80 Vezérlésátadás új adatok bevitelére.
- 200—210 Az adatbevitel vége, a file lezárása.

Abban az esetben, ha a NEVSOR nevű file már létezik, s a későbbiekben új embereket is fel szeretnénk venni a nyilvántartásba, akkor a file-t bővítésre kell megnyitni. A bővítőprogram általában csak az OPEN utasításban különbözik az elsődleges adatfelvivő programtól. Módosítsuk az előző programot a következőképpen:

```
10 OPEN1,8,2,"NEVSOR,S,A"
```

Ha újra futtatjuk, akkor a most beírt rekordokat hozzáfűzi a program a már meglevőkhöz, s így az állományt tetszés szerint bővíthetjük.

4.2.4 Olvasás soros file-ból

Egy file-ba beírt adatokat az INPUT és a GET BASIC utasításokkal olvashatjuk el. Ezek formátuma:

```

INPUT#lfn, változólista
GET#lfn, változó

```

ahol:

#lfn az OPEN utasításban megadott logikai file-szám.

Ezek az utasítások pontosan úgy működnek egy adott file-ra, mint azt a BASIC fejezetből már korábban megtanultuk. Tehát az INPUT utasítás a változó típusának megfelelő értéket beolvassa a billentyűkről, ha a RETURN billentyűt lenyomtuk (ez jelzi az adat végét). Az INPUT# utasítás beolvassa a megnyitott file-ból a változó típusának megfelelő adatot a RETURN kódjáig. A RETURN kódját az adat végére a PRINT# lfn utasítás tette. Nézzünk erre is egy példát!

Példa

Az előző fejezetben felírt NEVSOR nevű soros file-t olvassuk el!

```

5 REM 86. PROGRAM
10 OPEN1,8,2,"NEVSOR"
20 INPUT#1,N$
30 INPUT#1,EV
40 PRINT"NEV          : ";N$
50 PRINT"SZULETESI EV: ";EV
60 GOTO 20

```

Futtassuk le a programot! A program az előbb felírt adatokat kiírja ugyan a képernyőre, de csak a STOP billentyűre áll le. Ezt a hibát könnyű kijavítani, ha megkérdeztetjük a programmal, hogy hány rekordból áll a file. Írjuk be a programba a következő sorokat:

```

5 REM 87. PROGRAM
15 INPUT"REKORDOK SZAMA: ";DB
55 I=I+1
56 IF I=DB THEN 200
200 REM AZ OLVASAS VEGE
210 CLOSE1
220 STOP

```

Ha tudjuk azt, hogy a NEVSOR nevű soros állomány pl. három rekordot tartalmaz akkor a 15-ös sorban levő INPUT utasításra feleljük azt, hogy 3. Így a program csak három rekordot fog kiírni és automatikusan megáll. Azonban ez a megoldás kissé nehézkes és a gyakorlatban ritkán alkalmazzuk. Jó lenne egy olyan programot írni, amelyik jól működik akkor is, ha nem tudjuk előre, hogy hány rekord van a file-ban, hiszen a számítógép jelzi azt, ha az utolsó rekordot beolvastuk. Ezt az üzenetet (pontosabban az üzenet kódját) számunkra hozzáférhető formában az ST nevű rendszerváltozóba helyezi el. Miután ez rendszerváltozó, a neve is kötött tehát a BASIC programban más célra nem is használhatjuk az egyenlőségjel bal

oldalán. Ha értéket próbálunk adni, pl. ST=0, akkor a SYNTAX ERROR hiba-üzenetet kapjuk. Ez logikus is, hiszen értéket csak az operációs rendszer adhat az ST változónak, ezt mi csak leolvashatjuk. Alapesetben az ST változó értéke 0, ha az utolsó rekordot beolvastuk, akkor értéke 64 lesz. Ha az értéke nem 0 és nem 64, akkor az INPUT/OUTPUT művelet során fellépő valamilyen hibára utal a kód (pl., ha az utolsó rekordot már beolvastuk és még olvasunk a file-ból, akkor az ST=66 hibakódot kapjuk).

Az elmondottakat nézzük meg a gyakorlatban is! Ha már lefuttattuk az előző programot, akkor írassuk ki az ST változó értékét! Ha pontosan annyit adtunk meg az olvasandó rekordok számának, mint amennyi rekord a file-ban volt, akkor az ST változó értéke minden bizonnyal 64 lesz.

Írjuk be:

```
PRINT ST
64
```

Módosítsuk a programot úgy, hogy ne kelljen megadni a file-ban levő rekordok számát. Az 55-ös sort töröljük ki, természetesen a 15-ös sörrel együtt!

Módosítsuk az 56-os sort:

```
56 IF ST=64 THEN 200
```

Futtassuk le a programot, amely a következőkre módosult a javítások után:

```
5 REM 88. PROGRAM
10 OPEN1,8,2,"NEVSOR"
20 INPUT#1,N$
30 INPUT#1,EV
40 PRINT"NEV      : ";N$
50 PRINT"SZULETESI EV: ";EV
55 I=I+1
56 IF ST=64 THEN 200
60 GOTO 20
200 REM AZ OLVASAS VEGE
210 CLOSE1
220 STOP
```

4.2.5 A hibacsatorna leolvasása

A programozási munka elsajátításának kezdetén valószínűleg gyakran, később ritkábban, de előfordul, hogy valamilyen hibát vétünk. A hibák nem okoznak különösebb fejtörést, ha tudjuk azok okát, mert akkor lehet javítani. A lemezkezelés során már találkozhatunk olyan esetekkel, amikor a lemezegységen levő piros LED szaporán villogni kezdett. Eddig csak azt jegyeztük meg, hogy valamilyen hibát vétettünk. A hibákat részint szándékosan hoztuk létre, így könnyen kitalálhattuk az okot is. A lemezegység operációs rendszere nagyon sok hibát felismer, s annak okáról hibaüzenetet küld a programozónak, erről azonban az alapgép operációs rendszere nem tud tájékoztatni. Ahhoz, hogy a hiba okát megállapíthassuk, a 15-ös parancs-

csatornán keresztül le kell olvasnunk a lemezegység által küldött hibaüzenetet. Az olvasást az INPUT utasítással végezhetjük el. A hibaüzenet négy mezőből áll:

- a hiba kódja (numerikus);
- a hiba szöveges megnevezése (karakteres);
- a hiba előfordulási helyének sáv-száma (numerikus);
- a hiba előfordulási helyének szektora (numerikus).

Ha a hiba jellegéhez nem kapcsolható a lemezen sáv- és szektorszám (pl. SYNTAX ERROR esetén), akkor ezek 0—0 értéket vesznek fel.

Nézzünk egy példát a hibacsatorna leolvasására!

```
5 REM 89. PROGRAM
10 OPEN1,8,15
20 INPUT#1,H1,H2#,H3,H4
40 PRINT H1;H2#;H3;H4; :REM KEPERNYORE KIIRAS
50 CLOSE1
```

Futtassuk le a programot! Ha a lemezen levő piros LED előtt nem villogott, tehát minden rendben volt, s a program begépelésekor sem vétettünk hibát, akkor a következő üzenetet látjuk:

```
00, OK, 00, 00
```

Ez az üzenet azt jelenti, hogy a lemezegységen most minden rendben van. A sáv- és szektormező 00, mert ez az üzenet nem igényli ezeket az adatokat.

Idézzünk elő most egy lemezhibát! Pl.:

```
OPEN1,8,15,"S ADATOK"
CLOSE1
```

A parancsok begépelése után a lemezegységen levő piros LED szaporán villogni kezd. Futtassuk le a hibacsatorna-leolvasó programot! A képernyőn a következő üzenet fog megjelenni:

```
34, SYNTAX ERROR,00,00
```

Vagyis a lemezegységnek küldött parancsban az operációs rendszer formai hibát (SYNTAX ERROR) érzékelt. Hiányzik a kettőspont az S (SCRATCH) parancs után. Az előbbi hibacsatorna-leolvasó programot célszerű alprogram formájában minden programban alkalmazni a lemezparancsok kiadása után.

A lemezegység hibaüzeneteit és azok okait a *Függelékben* foglaltuk össze.

4.2.6 Soros adatfeldolgozás

Ebben a fejezetben egy teljes programot mutatunk be a soros adatállományok kezelésére. A feladat egy házi könyvtár nyilvántartó programjának készítése. A programnak legyen adatjavító, visszakereső és listázó része. Ezeket a funkciókat a felhasználó a menüből választhassa. Az adatokat lemezen egy soros file-ba írjuk. A nyilvántartásban szerepeljen a könyv címe, szerzője és egy rövid megjegyzés a könyv témájáról. Ezek az adatfajták alkossák a rekordokat. A könnyebb kezelhetőség kedvéért fel-

veszünk még egy sorszámmezőt is. A sorszámot a program határozza meg, s célja az egyedi nyilvántartás.

Mielőtt a program írásához hozzákezdénénk, gondoljuk végig, hogy az milyen főbb részekből építhető fel!

Vannak ui. olyan általános eljárások, amelyek szinte minden adatfeldolgozási feladatban előfordulnak. Ezeket célszerű előre alprogram formájában megírni, így más rendszerekhez is könnyen alkalmazhatók lesznek. A feladat kidolgozása során mi is ezt a szemléletet követtük.

a) A menü

Ezzel a képernyőn megjelenő szöveggel indul a program. Feladata, hogy rövid, tömör szöveges tájékoztatást adjon a program feladatáról, s egyben funkcióiról a felhasználónak. A program feladatára itt a program címe utal: KÖNYVTÁRKEZELÉS.

A funkciók:

1. ADATBEVITEL
2. JAVITAS
3. KERESÉS
5. LISTAZAS
5. PROGRAM VEGE

A funkciók közül a megfelelő számjegybillentyű lenyomásával választhatunk.

Feladatunk tehát az, hogy esztétikus, könnyen áttekinthető formában a fenti menüt megjelenítsük. Ehhez szükségünk lesz egy olyan alprogramra, amely a kurzort a képernyő kívánt helyére pozicionálja. Ezt a következőképpen írtuk meg:

```
2000 REM A KURZOR POZICIONALASA
2010 POKE 214,SR
2020 POKE 211,OSZ
2030 SYS58640
2040 RETURN
```

A változók jelentése:

SR — a kurzor kívánt sor értéke;
OSZ — a kurzor kívánt oszlop értéke.

A sor- és oszlopkoordináták alapján az adott pozícióhoz tartozó képernyőtár-, ill. szintár címet az operációs rendszer 58640-es címen kezdődő rutinjával végeztettük el.

A megjelenített képet korlátlan ideig tartani kell, ill. a menüpontok választásához is egy billentyű lenyomására kell várni. Célszerűnek látszik egy olyan alprogram készítése, amely vár egy tetszőlegesen lenyomott billentyűre.

```
3000 REM VARAKOZAS EGY BILLENTYURE
3010 :
3020 A$=""
3030 GET A$: IF A$="" THEN 3030
3040 RETURN
```

A programrészlet az A\$ változóba visszaadja a lenyomott billentyű karakterét. E két rutin felhasználásával a program menüjét kiíró programrészlet a következő:

```
10 REM BEJELENTKEZES
13 GOSUB 10000
14 MD=15
15 OPEN15,8,15
16 DIM C$(MD),SZ$(MD),TM$(MD)
20 :
30 PRINT" "
40 POKE53280,12:POKE53281,12:PRINT"■";
50 SR=1:OSZ=8:GOSUB2000
60 PRINT" "
65 SR=SR+1:GOSUB 2000
70 PRINT"| KONYVATAR KEZELES |"
75 SR=SR+1:GOSUB 2000
80 PRINT" "
85 GOSUB 500
90 SR=SR+2:GOSUB 2000
100 PRINT"      M E N U "
110 SR=SR+4:OSZ=8:GOSUB 2000
120 PRINT"1■ ADATBEVITEL"
130 SR=SR+2 : GOSUB 2000
140 PRINT"2■ JAVITAS"
150 SR=SR+2 : GOSUB 2000
160 PRINT"3■ KERESÉS"
170 SR=SR+2 : GOSUB 2000
180 PRINT"4■ LISTAZAS"
190 SR=SR+2 : GOSUB 2000
192 PRINT"5■ PROGRAM VEGE"
190 SR=SR+3:OSZ=4:GOSUB 2000
200 PRINT"KEREM VALASSZON....."
210 GOSUB 3000
220 A=VAL(A$)
230 IF A<1 OR A>5 THEN GOSUB 3700:GOTO 30
240 ON A GOSUB 4000,5000,6000,7000,250
245 GOTO 30
250 PRINT"2000 A PROGRAM FUTASA BEFEJEZODOTT"
260 CLOSE15:STOP
```

- 30 A képernyő törlése.
- 40 A szürke háttér és a keretszín beállítása, fekete betűkkel való íráshoz.
- 50 Az 1. sor 8. oszlopba pozicionáljuk a kurzort.
- 60—80 A program címének kiírása egy keretben.
- 90—190 Az egyes menüpontok kiírása.
- 200—230 A választás beolvasása ellenőrzéssel.
- 240 A vezérlés átadása a választott menüpontra.

b) Hibáüzenet kiírása

Ha a menüpont választásánál hibát vétünk, akkor a következő program tájékoztat a hiba okáról.

```
3500 REM HIBAUZENET KIIRASA
3510 :
3520 A$="":OSZ=8:GOSUB 2000
3540 SR=23:GOSUB 2000
3550 PRINT"■";
3560 PRINTH$
3570 FOR ZX=1TO1000:NEXT
3575 GET A$:IF A$<>" "THEN 3630
3580 PRINT"■";
3590 GOSUB 2000
3600 PRINT"RETURN HA JAVITJA. "
3610 FOR ZX=1TO1000:NEXT
3615 GET A$:IF A$<>" "THEN 3630
3620 GOTO3540
3630 GOSUB 2000
3640 PRINT"■"
3650 RETURN
```

A program a kiírt hibáüzenetet a figyelem felkeltésére villogtatja, amíg a kezelő azt észreveszi, elolvassa. A RETURN billentyű lenyomása után a választást meg lehet ismételni. A hibáüzenet a H\$ változóban van. Az üzenet vibrálásának sebességét a 3570-es, ill. a 3610-es sorokban levő üres ciklussal állíthatjuk be. Az üzenet a következőképpen hívható:

```
3700 H$="A BEIRT SZAM HIBAS"
3710 GOSUB 3500
3720 RETURN
```

c) A képernyőmaszk

Amint azt az előző fejezetben láttuk, a képernyőmaszk nagyon fontos a rekord mezőinek esztétikus megjelenítésében. Szinte minden menüpontban szükségünk lesz a mezők elnevezéseinek kiírására. Erre való a következő alprogram:

```
3200 REM KEPERNYO MASZK ADATBEVITELHEZ
3210 :
3230 SR=8:OSZ=3:GOSUB 2000
3240 PRINT"SORSZAM:"
3250 SR=11:OSZ=3:GOSUB 2000
3260 PRINT" CIM:"
3270 SR=14:OSZ=3:GOSUB 2000
3280 PRINT" SZERZO:"
3290 SR=17:OSZ=3:GOSUB 2000
3300 PRINT"TEMAKOR:"
3370 RETURN
```

A programrészlet nem tartalmaz új eljárást, így ennek értelmezését az olvasóra bízuk.

d) Az adathalmaz beolvasása

Az adatokat a lemezen a SZEKRENY nevű soros file-ban tároljuk, s minden esetben innen olvassuk be. Amennyiben a file még nem létezik, úgy természetesen semmit sem tudunk beolvasni. Ha azonban a file már létezik, akkor a program automatikusan olvassa be az összes rekordot a tárba, megjegyezve a rekordok számát. Ezt végzi el a következő alprogram.

```
500 REM LEMEZROL BEOLVASAS
510 :
512 IF BE<>0 THEN RETURN
513 BE=1
520 OPEN1,8,2,"SZEKRENY"
530 GOSUB 3400
540 IF H1<>0 THEN JS=0:RETURN
550 JS=JS+1
560 INPUT#1,C$(JS)
570 INPUT#1,SZ$(JS)
580 INPUT#1,TM$(JS)
590 IF ST<>0 THEN CLOSE1:RETURN
600 GOTO 550
```

Az 520-as sorban megnyitjuk a SZEKRENY nevű soros file-t olvasásra, majd leolvassuk a hibacsatornát (530), hogy a megnyitás eredményes volt-e. A hibacsatorna leolvasását a következő alprogrammal végezhetjük el.

```
3400 REM HIBACSATORNA LEOLVASAS
3410 :
3420 INPUT#15,H1,H2$,H3,H4
3430 IF H1=0 THEN RETURN
3440 IF H1=62 THEN CLOSE1:GOTO3470
3450 H$=H2$
3460 GOSUB 3500
3470 PRINT#15,"I"
3480 RETURN
```

Amennyiben a file már létezett, s egyéb lemezhiba sem fordult elő, úgy H1 értéke 0 lesz. Ha a file még nem létezik, akkor H1=62 lesz. Ebben az esetben a beolvasást nem végezzük el (l. 540-es sor). Más lemezhiba esetén a leolvasott hibáüzenetet kiírjuk (3450-es, 3460-as), majd a lemezt inicializáljuk (alapállapotba hozzuk), hogy a páros LED ne villogjon (3470-es). Ha a file létezett, úgy a rekordokat a file végéig beolvassuk (550—590-es).

Az előzőekben bemutatott programrészleteket, alprogramokat más adatfeldolgozó programoknál is hatékonyan alkalmazhatjuk, hiszen ilyen jellegű problémák mint pl. hibacsatorna leolvasás, kurzor pozicionálás, gyakran előfordulnak.

A továbbiakban az egyes menüprogramokat megvalósító programrészleteket mutatjuk be.

e) Adatbevitel

E programrésszel tölthetjük fel az adatállományunkat adatokkal, ill., ha már az adatállomány létezik, akkor bővíthetjük azt. Az adatbevitelt a képernyőmaszk rutinnal végezzük el.

```

4000 REM ADATBEVITEL
4010 :
4012 JS=JS+1
4015 PRINT "J":POKE53280,9:POKE53281,9:PRINT "J"
4016 SR=2:OSZ=8:GOSUB 2000
4017 PRINT " "
4018 SR=SR+1:GOSUB 2000
4019 PRINT "I      ADATBEVITEL      I "
4020 SR=SR+1:GOSUB 2000
4021 PRINT " "
4025 GOSUB 3200
4026 GOSUB 3200
4030 :
4040 SR=8:OSZ=12:GOSUB 2000
4050 PRINT "J";JS;"J"
4060 SR=11:OSZ=11:GOSUB 2000
4070 INPUT C$(JS)
4072 IF C$(JS)="* THEN C$(JS)="*"
4073 IF C$(JS)="VEGE" THEN 4200
4080 SR=14:OSZ=11:GOSUB 2000
4090 INPUT SZ$(JS)
4092 IF SZ$(JS)="* THEN SZ$(JS)="*"
4100 SR=17:OSZ=11:GOSUB 2000
4110 INPUT TM$(JS)
4112 IF TM$(JS)="* THEN TM$(JS)="*"
4120 JS=JS+1
4130 GOTO 4040
4200 JS=JS-1
4210 REM AZ ADATHALMAZ MENTESE
4212 PRINT "J";JS;"J"
4213 PRINT "J" KEREM AZ ADATLEMEZT A 8-AS MEGHAJTABA
4214 PRINT "J" RENDBEN CTRL
4215 WAIT 653,4
4220 OPEN 1,8,2,"@:SZEKRENY,S,W"
4230 FOR I=1 TO JS
4240 PRINT#1,C$(I)
4250 PRINT#1,SZ$(I)
4260 PRINT#1,TM$(I)
4265 SR=15:OSZ=2:GOSUB 2000
4266 PRINT "A LEMEZRE IRT REKORDOK SZAMA:";I
4270 NEXT
4280 CLOSE 1
4998 GOSUB 3000
4999 RETURN

```

4210—4270 A beírt adatokkal lemezre mentjük a régi állomány felülírásával.

Az üresen hagyott mezőket * karakterrel töltjük ki, így ha a lemezegységről INPUT-tal olvasunk, akkor az üres karakterre való várakozási állapotba kerülést elkerülhetjük.

f) Javítás

A bevitt adatok javítását a sorszám alapján végezhetjük el. A beírt sorszám alapján a képernyőn megjelenik a hozzá tartozó rekord, s az egyes mezőket felülírással módosíthatjuk. Amennyiben a beírt sorszám nagyobb, mint a nyilvántartásban szereplő rekordok száma, úgy az utolsó rekordot kapjuk meg javításra. Értelemszerűen, ha 1-nél kisebb sorszámot írunk be, akkor a program az első rekordra áll. Ha nulla sorszámot írunk be, akkor a program a javított adatokat lemezre írja. Az aktuális sorszám értékét a J változóban tároljuk.

5100—5200 Az új adatok beolvasása.

5210—5250 Az előző rekord képernyőmezőinek törlése.

```

5000 REM JAVITAS
5005 REM JS=JS-1
5010 :
5015 PRINT "J":POKE53280,6:POKE53281,6:PRINT "J"
5016 SR=2:OSZ=8:GOSUB 2000
5017 PRINT " "
5018 SR=SR+1:GOSUB 2000
5019 PRINT "I      J A V I T A S      I "
5020 SR=SR+1:GOSUB 2000
5021 PRINT " "
5022 GOSUB 3220
5023 OPEN 4,3
5024 SR=22:GOSUB 2000
5025 PRINT "J";VEGE;"J" SORSZAM=0
5030 GOSUB 3200
5040 SR=8:OSZ=12:GOSUB 2000
5050 INPUT "J";J
5055 IF J=0 THEN CLOSE 4:GOTO 5300
5060 IF J<1 THEN J=1
5070 IF J>JS THEN J=JS
5080 GOSUB 3000
5100 SR=11:OSZ=11:GOSUB 2000
5120 INPUT "J";C$(J)
5130 IF C$(J)="* THEN C$(J)="*"
5140 SR=14:OSZ=11:GOSUB 2000
5150 INPUT "J";SZ$(J)
5160 IF SZ$(J)="* THEN SZ$(J)="*"
5170 SR=17:OSZ=11:GOSUB 2000
5180 INPUT "J";TM$(J)
5190 IF TM$(J)="* THEN TM$(J)="*"
5200 SR=11:OSZ=11:GOSUB 2000
5210 PRINT " "
5211 SR=8:OSZ=11:GOSUB 2000
5212 PRINT " "
5220 SR=14:OSZ=11:GOSUB 2000
5230 PRINT " "
5240 SR=17:OSZ=11:GOSUB 2000
5250 PRINT " "
5260 GOTO 5040
5300 GOSUB 4210
5310 RETURN
5998 GOSUB 3000
5999 RETURN

```


g) Listázás

A nyilvántartásban levő adatok listáját kérhetjük képernyőre, nyomtatóra (7025—7090-es). A +/- billentyűvel tetszés szerint lapozhatunk az adatállományban.

```

7000 REM LISTAZAS
7010 :
7015 PRINT "┐":POKE53280,5:POKE53281,5:PRINT"┐"
7016 SR=2:OSZ=8:GOSUB2000
7017 PRINT"┐"
7018 SR=SR+1:GOSUB 2000
7019 PRINT"┐ L I S T A Z A S ┐"
7020 SR=SR+1:GOSUB 2000
7021 PRINT"┐"
7025 SR=10:GOSUB 2000
7030 PRINT"KEPERNYORE.....3"
7035 SR=SR+2:GOSUB 2000
7040 PRINT"NYOMTATORA.....4"
7050 SR=SR+3:GOSUB 2000
7060 PRINT"KEREM VALASSZON...."
7070 GOSUB 3000
7080 A=VAL(A$)
7090 IF A<>3 AND A<>4 THEN GOSUB 3700:GOTO7000
7100 OPEN4,A
7110 SR=10:GOSUB 2000
7120 PRINT"┐"
7130 SR=SR+2:GOSUB 2000
7140 PRINT"┐"
7150 SR=SR+3:GOSUB 2000
7160 PRINT"┐"
7170 GOSUB 3200
7200 SR=22:OSZ=3:GOSUB 2000
7210 PRINT"LAPOZAS: ┐+┐ ELORE ┐-┐ HATRA"
7220 SR=23:OSZ=12:GOSUB 2000
7230 PRINT"┐┐┐ VISSZA A MENURE"
7240 :
7250 J=1
7260 GOSUB 8000
7310 GOSUB 3000
7320 IF A$="+"THEN J=J+1:GOTO 7400
7330 IF A$="-"THEN J=J-1:GOTO 7400
7340 IF A$="V"THEN 7430
7350 GOTO 7310
7400 IF J<1 THEN J=1
7410 IF J>JS THEN J=JS
7411 GOSUB 8100
7420 GOTO 7260
7430 CLOSE4
7440 RETURN
7998 GOSUB 3000
7999 RETURN

```

h) Keresés

A tetszőleges szempont szerint keresésnél a feladat a következő: egy változó hosszúságú füzérben meg kell vizsgálni, hogy egy másik füzér előfordul-e, s ha igen, meg kell határozni az első előfordulás helyét (l. a 2.3. alfejezetben). Pl.: A\$="ABCDE" füzérben a B\$="CD" megtalálható a 3. pozíciótól. Ezt a következő BASIC program-részlettel valósíthatjuk meg.

```

5 REM 91. PROGRAM
10 INPUT A$
20 INPUT B$
30 FOR I=1 TO LEN(A$)
40 IF MID$(A$,I,LEN(B$))=B$ THEN 70
50 NEXT I
60 I=0
70 PRINT I

```

Tehát a B\$ füzér első előfordulási helyét kerestetjük meg a A\$-ban. Az I változó értéke nulla, ha a B\$ nem található meg az A\$-ban. Egyébként a program az első előfordulási helyet adja vissza. Keresőprogramként a bemutatott alprogram helyett annak gépi kódú változatát használjuk a közölt programban, ui. nagy tömegű adatok esetén ez a BASIC változat lassan futna le.

```

10000 REM !POS FUGGVENY
10010 :
10020 :
10030 FOR I=51200 TO 51440
10040 READ X:POKE I,X:S=S+X:NEXT
10050 DATA 169, 11,160,200,141, 10, 3,140, 11, 3, 96,169
10060 DATA 0,133, 13, 32,115, 0,201, 33,240, 6, 32,121
10070 DATA 0, 76,141,174, 32,115, 0,201,185,240, 10,201
10080 DATA 196,208, 3, 76,177,200, 76, 8,175, 32,115, 0
10090 DATA 32,250,174, 32,158,173, 32,143,173,165,100, 72
10100 DATA 165,101, 72, 32,253,174, 32,158,173, 32,163,182
10110 DATA 240,100,133, 4,134,251,132,252,104,168,104, 32
10120 DATA 170,182,240, 86,133, 3,134,253,132,254,162, 0
10130 DATA 32,121, 0,201, 44,208, 7, 32,155,183,138,240
10140 DATA 65,202,134, 6, 32,247,174,165, 3, 56,229, 4
10150 DATA 144, 40,105, 0,133, 5,165, 6, 24,101,253,133
10160 DATA 253,144, 2,230,254,160, 0,177,251,209,253,208
10170 DATA 11,200,196, 4,144,245,164, 6,200, 76,162,179
10180 DATA 230, 6,198, 5,208, 4,160, 0,240,243,230,253
10190 DATA 208,223,230,254,208,219, 76, 72,178, 32,115, 0
10200 DATA 32,250,174, 32,158,183,138, 72, 32,253,174, 32
10210 DATA 158,173, 36, 13, 48, 12, 32,170,177,165,100,208
10220 DATA 225,165,101, 76,219,200, 32,130,183,240,215,160
10230 DATA 0,177, 34,133, 3,104, 32,125,180,168,240, 7
10240 DATA 165, 3,136,145, 98,208,251, 32,202,180, 76,247
10250 DATA 174
10260 IF S<>30119 THEN PRINT "HIBA A DATA SORBAN..!!"
10270 SYS 51200
10280 RETURN

```


Mivel a gépi kódú programozással csak a 6. fejezetben foglalkozunk, ezért a programot DATA sorokban, decimális számok formájában közöljük (10050—10250-es).

Az 10030—10040-es sorokban levő ciklussal töltjük a programot a ciklusváltozóval kijelölt tárcímtartományba. Ez az alprogram önállóan is használható. Kiegészíti a BASIC utasításkészletet !POS nevű függvénnyel, amelynek feladata, hogy egy füzérben a másik füzér első előfordulási helyét visszaadja. Más BASIC-beli függvényekhez hasonlóan hívható. Szintaktikája a következő:

I=!POS (A\$,B\$,P)

ahol A\$ az a füzér, amiben a B\$-t keressük, P pedig az a pozíció, ahonnan a keresést elkezdjük. Az I változó értéke 0, ha a B\$ nem fordul elő az A\$-ban, egyébként pedig az első előfordulás pozíciója. Az előzőek alapján a kereső programrész szerkezete könnyen áttekinthető.

```
6000 REM KERESÉS
6005 CLOSE#4: OPEN 4,3
6010 :
6015 PRINT "Ú":POKE53280,7:POKE53281,7:PRINT "■"
6016 SR=2:OSZ=8:GOSUB 2000
6017 PRINT " "
6018 SR=SR+1:GOSUB 2000
6019 PRINT " | K E R E S E S | "
6020 SR=SR+1:GOSUB 2000
6021 PRINT " "
6030 GOSUB 3000
6040 GOSUB 8000
6050 J=0:C$="":SZ$="":TM$="":GOSUB 8000
6060 SR=8:OSZ=12:GOSUB 2000
6070 PRINTJ
6080 GOSUB 2000
6090 INPUT "■";J
6095 IF J=0 THEN 6110
6100 IF J>0 AND J<=JS THEN GOSUB 8000:GOSUB 3000:GOTO 6050
6105 GOTO 6050
6110 SR=11:GOSUB 2000
6120 INPUT "■";C$
6130 SR=14:GOSUB 2000
6140 INPUT "■";SZ$
6150 SR=17:GOSUB 2000
6160 INPUT "■";TM$
6200 REM KERESÉS
6210 JT=1
6220 IF LEN(C$)=0 OR C$="*" THEN 6300
6230 FOR I=JT TO JS
6240 IF !POS(C$(I),C$,1)<>0 THEN JT=I:GOTO 6300
6250 NEXT
6260 GOSUB 8500
6270 GOTO 6220
6300 IF LEN(SZ$)=0 OR C$="*" THEN 6400
6310 FOR I=JT TO JS
6320 IF !POS(SZ$(I),SZ$,1)<>0 THEN JT=I:GOTO 6400
6330 NEXT
```

```
6340 GOSUB 8500
6350 GOTO 6998
6400 IF LEN(TM$)=0 OR C$="*" THEN 6500
6410 FOR I=JT TO JS
6420 IF !POS(TM$(I),TM$,1)<>0 THEN JT=I:GOTO 6500
6430 NEXT
6440 GOSUB 8500
6450 GOTO 6998
6500 J=JT:GOSUB 8000
6505 JT=JT+1
6510 GOSUB 3000
6520 IF JT<=JS THEN GOTO 6220
6530 GOTO 6998
6998 GOSUB 3000
6999 RETURN
```

i) Vegyes alprogramok

A programlisták átnézése közben találkozhattunk olyan alprogram-hivatkozásokkal, amelyeket még nem ismertettük. Itt most áttekinthetjük ezek működését. Külön alprogramban való megírásuk azért volt célszerű, mert menüpontban is előfordulnak, esetleg egy menüpontban többször is.

```
8500 SR=22:OSZ=3:GOSUB 2000
8510 PRINT "A REKORD A NYILVANTARTASBAN"
8520 SR=22:OSZ=3:GOSUB 2000
8530 PRINT "NEM SZEREPEL"
8540 GOSUB 3000
8550 SR=22:OSZ=3:GOSUB 2000
8560 PRINT " "
8570 SR=22:OSZ=3:GOSUB 2000
8580 PRINT " "
8590 RETURN
```

A rutin egy hibaüzenetet ír a képernyőre, ha a keresett rekordból több nem szerepel a nyilvántartásban.

A következő alprogram a kezelőnek szóló információs üzenetet írja a képernyőre.

```
8200 SR=22:GOSUB 2000
8210 PRINT "VEGE: CIM=VEGE BEIRASRA"
8220 SR=23:GOSUB 2000
8230 PRINT "KITOLTETLEN MEZO"
8240 RETURN
```


A következő két alprogram feladata egy rekord kiírása, ill. törlése a képernyőről.

```
8000 REM J-EDIK REKORD KIIRASA
8005 :
8010 SR=8:OSZ=12:GOSUB 2000
8020 PRINT#4,J
8030 SR=11:OSZ=12:GOSUB 2000
8040 PRINT#4,C#(J)
8050 SR=14:OSZ=12:GOSUB 2000
8060 PRINT#4,SZ#(J)
8070 SR=17:OSZ=12:GOSUB 2000
8080 PRINT#4,TM#(J)
8090 RETURN
```

```
8100 REM J-EDIK REKORD TORLESE
8105 :
8110 SR=11:OSZ=12:GOSUB 2000
8120 PRINT#4,"
8130 SR=14:OSZ=12:GOSUB 2000
8140 PRINT#4,"
8150 SR=17:OSZ=12:GOSUB 2000
8160 PRINT#4,"
8170 RETURN
```

A könyvtárkezelő program jobb áttekinthetősége kedvéért a felhasznált alprogramokról egy táblázatos összefoglalót is közreadunk.

<i>A programsor száma</i>	<i>Az alprogram feladata</i>
10000—10280	A !POS fv. gépi kódú programja DATA sorokban.
8500—8590	"TÖBB REKORD A NYILVÁNTARTÁSBAN NEM SZEREPEL" üzenet kiírása.
8200—8240	Kezelői üzenetek kiírása.
8100—8170	A J-edik rekord törlése.
8000—8090	A J-edik rekord kiírása képernyőre.
7000—7999	LISTÁZÁS menüpont.
6000—6999	KERESÉS menüpont.
5000—5999	JAVÍTÁS menüpont.
4000—4999	ADATBEVITEL menüpont.
3500—3650	Hibaüzenet kiírása.
3400—3480	Hibacsatorna leolvasása.
3000—3040	Várakozás egy billentyű lenyomására.
2000—2040	A kurzor pozicionálása.
500—600	Az adathalmaz lemezről való beolvasása.
30—245	A menü képernyőre való kiírása.

4.3 A RELATÍV FILE SZERVEZÉSE

Az előző pontban tárgyaltuk, hogy soros adattárolás esetén az egyes rekordokat csak a felírás sorrendjében lehet beolvasni a háttértárolóról. Sok adatfeldolgozási feladat esetében ez a módszer jól alkalmazható, különösen akkor, ha az adathalmaz a feldolgozói programmal együtt elfér a tárban.

Ha azonban az adathalmaz túl nagy, s nem fér be az operatív tárba, akkor a soros file-szervezés lényeges korlátozást jelent az adatfeldolgozás során. A relatív file-szervezés azonban ebben az esetben is kitűnően alkalmazható. Egyik nagy előnye a soros file-szervezéssel szemben az, hogy az adatokat tetszőleges sorrendben is be tudja olvasni a háttértárolóról.

Ismét a zenei példát véve, a relatív file analóg egy hanglemezen levő zeneszámok halmazával, s az egyes rekordok itt is megfeleltethetők az egyes zenedaraboknak. Tehát, ha egy meghatározott zeneszámmra vagyunk kíváncsiak, akkor a lemezjátszó hangszedőjét a kívánt szám fölé pozicionáljuk anélkül, hogy az összes előtte levőt meghallgatnánk.

A relatív file esetén is pontosan így járhatunk el, azzal a különbséggel, hogy a pozicionálás bonyolult műveletét is a számítógépre, pontosabban a lemezmeghajtó egység operációs rendszerére bizzuk. A beolvasandó rekord sorszámának közlése után az operációs rendszer (DOS) vezérli a fejmozgató mechanikát a kérdéses rekordhoz. Ez az adatelérési módszer meghatározza az egység fizikai megvalósítását is. Tehát a relatív file-kezelést csak lemezegységeken lehet alkalmazni, szalagos egységeken értelemszerűen nem. Ez az egyik hátránya a soros adattárolással szemben, hiszen azt az elvet változtatás nélkül alkalmazhatjuk szalag- és lemezegységen egyaránt. A másik hátránya talán az, hogy ez a forma egy kicsit bonyolultabb, mint a soros adatkezelés, azonban a gyorsabb működés, a nagyobb tömegű adatkezelési lehetőség bőven kárpótolt ezért a kis „kellemetlenségért”.

A soros file esetében a név már utal az adattárolás módjára. De mire utal a „relatív” jelző? Erre a REL-file szerkezete ad választ.

A REL-file azonos hosszúságú rekordok sokaságából áll, amelyeket a file létrehozásakor előre lefoglalunk. Így a felhasználó szabadon hozzáférhet bármelyik rekordhoz, függetlenül attól, hogy olvasni vagy írni akar a file-ba.

A programnak nem is kell tudnia, hogy a keresett rekord a lemezen fizikailag hol van. A lemezegység operációs rendszere, a DOS ún. rekordmutatókat képez, s ezeken keresztül „talál rá” fizikailag a kívánt rekordra. Tehát, ha közöljük a lemezegységgel, hogy a tizedik rekordot szeretnénk elérni, akkor ezt a sorszámot a DOS a file kezdetétől számított tizedik helyként, tehát *relatív sorszámként* értelmezi. A továbbiakban a rekord sorszámát kulcsnak nevezzük. E relatív sorszámból képzí a DOS a fejmechanika pozicionálásához szükséges adatokat, s így áll a kiválasztott rekordra. Attól függően, hogy a következő utasítás PRINT# vagy INPUT#, a számítógép felírja vagy elolvassa az adott rekordot. Ebből már láthatjuk, hogy a REL-file egyidejűleg mindkét hozzáférési mód (írás, olvasás) számára nyitott. Az OPEN utasításban nem kell külön meghatározni, hogy írni akarunk a file-ba, vagy olvasni akarunk belőle.

A relatív file kezelésének főbb lépései

a) A relatív file létrehozása:

1. a file megnyitása;
2. az utolsó rekordra való pozicionálás;
3. írás az utolsó rekordba (ekkor az előtte levő összes rekordot feltölti üres karakterrel);
4. a file lezárása.

b) Írás a relatív file-be:

1. a file megnyitása;
2. pozicionálás a kívánt rekordra;
3. a rekord kiírása;
4. a file lezárása.

c) Olvasás a relatív file-ból:

1. a file megnyitása;
2. pozicionálás a kívánt rekordra;
3. a rekord beolvasása;
4. a file lezárása.

A REL file-t tehát az ismertett három módon használhatjuk. Ha figyelmesen megnézzük, akkor mindhárom pontban az 1, 2, 4, alpontok megegyeznek, csak a 3. alpont az, amiben különböznek. Ezt az egyezést majd a programok szerkezete is tükrözni fogja.

4.3.1 A relatív file megnyitása

A soros file-okhoz hasonlóan a relatív file-okat is az OPEN utasítással nyitjuk meg. Az OPEN utasítás formátuma:

OPEN *lfsz*, 8, *mc*, "file-név, L," + CHR\$(*RH*)

ahol:

lfsz a logikai file-szám. Értékhátára megegyezik a soros file-nál tanultakkal;

8 a lemezegység fizikai száma;

mc másodlagos cím. Funkcióját tekintve megegyezik a soros file-nál elmondottakkal;

file-név rendeltetése ua., mint soros file-nál;

L jelzi a DOS számára, hogy egy relatív file-t nyitottunk meg;

CHR\$(*RH*) az *RH* a file-ban lévő rekordok hossza. A rekordhossz 1 és 254 között lehet. Ha a rekordhossz kisebb 88-nál akkor az egy INPUT# utasítással beolvasható.

Pl.: OPEN 1,8,2,"ADAT2,L"+CHR\$(81)

A 8-as lemezegységre megnyitunk egy ADAT2 nevű relatív file-t. Egy rekord max. hossza 80 karakter lehet. Egy byte-nyi hely ui. a rekordokat elválasztó RETURN kódjának kell. Tehát *RH* a rekord hasznos hossza + 1.

A rendelkezésre álló csatornák korlátozott száma miatt egy lemezegységen egyszerre csak egy REL file-t nyithatunk meg.

4.3.2 A relatív file lezárása

Ugyanúgy járunk el, mint a soros file lezárásakor, azonban a pozicionáláshoz szükségünk volt a 15-ös parancscsatornára (1. ott), tehát ezt is le kell zárni. A CLOSE és OPEN utasításokban a logikai file-számoknak természetesen meg kell egyeznie.

4.3.3 Pozicionálás a relatív file-ban

A pozicionálás a relatív file-kezelés legfontosabb része. Fizikailag is lezajlik, hiszen a DOS a rekord kulcsából meghatározza azt a pozíciót, ahol a rekord a lemezen fizikailag elhelyezkedik, s az író-olvasófejezet az adott lemezterületre irányítja. A pozicionáló utasítás formátuma:

PRINT# *lfsz*, "P" + CHR\$(*mc*) + CHR\$(*LB*) + CHR\$(*HB*) + CHR\$(*POZ*)

ahol:

lfsz
"P"

CHR\$(*mc*) a parancscsatorna logikai file-száma;
közöljük a DOS-sal, hogy REL-file pozicionálására kap parancsot;
a REL-file OPEN utasításában szereplő másodlagos cím (az adatcsatorna megnevezése);

CHR\$(*LB*) a rekordkulcsból meghatározott paraméterek. Miután az egy byte-ban ábrázolható legnagyobb szám 255, így ennél nagyobb kulcsot csak két byte-ban tudunk megadni. Ez a két byte (*LB* és *HB*) a következő képlet szerint számolható ki.

$HB = \text{INT}(\text{RN}/256)$
 $LB = \text{RN} - \text{HB} * 256$

ahol:

HB (High Byte) felső byte;
LB (Low Byte) alsó byte;
RN (Record Number) a rekord kulcsa, sorszáma.

Vagyis a tényleges rekord kulcsát (*RN*) egy két byte-os 256-os alapú számrendszerbeli számként közöljük a DOS-al. Ebből könnyen kiszámítható, hogy egy relatív file elvileg 65535 rekordot tartalmazhat (ha mindkét byte csak egyeseket tartalmaz, hexadecimálisan ez FFFF);

CHR\$(*POZ*) a kívánt kulcsú rekordon belüli pozíciószám. Ha a rekordot elejétől akarjuk beolvasni, akkor ide 1-et kell írni.

Pl.:

PRINT#15,"P"+CHR\$(2)+CHR\$(10)+CHR\$(0)+CHR\$(1)

A 2-es adatcsatornával jelölt REL-file 10. rekordjának első byte-jára pozicionálunk


```
PRINT#15,"P"+CHR$(2)+CHR$(10)+CHR$(1)+CHR$(3)
```

Pozicionálunk az LB=10 HB=1, vagyis $1 \times 256 + 10 = 266$. rekord 3. byte-jára a REL-file-on belül.

Emlékeztetől megjegyezzük, hogy a lemezegységhez küldött parancsokat ugyanúgy karakteres formában kell a DOS-nak elküldeni, mint az adatokat, csak a 15-ös másodlagos címmel jelölt csatornán keresztül. Az itt beérkező karaktersorozatot a DOS parancsként értelmezi. Amennyiben ez számára értelmes, úgy végrehajtja, ha értelmetlen, akkor a SYNTAX ERROR hibaüzenetet küldi és a lemezegységen levő piros LED is villog. A lemezparancsok karaktersorozatként való küldését pedig az indokolja, hogy az alapgép operációs rendszeréből azok hiányoznak. Ha a parancsot idézőjel nélkül írjuk a programba, akkor azokat az alapgép operációs rendszere próbálja megfejteni, s az mint tudjuk, számára sajnos értelmetlen.

4.3.4 A relatív file létrehozása

A relatív file létrehozásakor célszerű a teljes fileterületet „előkészíteni”, mert így a későbbi feldolgozás sokkal gyorsabbá válik. Az előkészítés azt jelenti, hogy a file legnagyobb kulcsú rekordjába CHR\$(255) karaktert írunk. Ha olyan rekord olvasásával próbálkozunk, amelynek kulcsa nagyobb a file eddigi utolsó rekordjának kulcsánál, akkor ez a

RECORD NOT PRESENT

hibához vezet. Ha azonban ugyanilyen kulcsú rekordot írunk a file-ba, akkor a DOS az új rekord kulcsánál kisebb kulcsú összes rekordot automatikusan teleírja CHR\$(255) karakterekkel. Így az új tartomány rekordjainak további olvasása már hibamentesen lehetséges.

Ha a feldolgozás során nem lépünk ki az előre létrehozott (kialakított) területből, akkor az egyes rekordok elérése is gyors. Ellenkező esetben, ha mindig túllépjük a max. kulcsot, s a file területének ismételt kialakítása mindig automatikusan megtörténik, akkor a REL-file kezelése igen lelassul.

Célszerű egy olyan programot írni, mely a REL-file által kitöltött területet előre lefoglalja a lemezen, az egyes rekordok szerkezetét kialakítja, s az első rekordba CHR\$(255) karaktert helyez. Tehát megjelöl lemezen egy „üres” REL-file-t.

```
5 REM 92. PROGRAM
100 I4=500 :REM MAXIMALIS REKORSZAM
110 N9=8 :REM A LEMEZEGYSEG FIZIKAI SZAMA
120 N1$="REL.DATA":REM A REL-FILE NEVE A LEMEZEN
130 :
140 REM REL.KREALO*****
150 :
160 GOSUB 220 :REM RENYIT----->
170 GOSUB 350 :REM RELPOINT----->
180 PRINT#2,CHR$(255)
190 CLOSE2:CLOSE15
200 STOP
210 :
```

```
220 REM *** REL.FILE MEGNYITASA ***
230 :
240 OPEN2,N9,2,N1$+",L,"+CHR$(32)
250 GOSUB 280:REM HIBA
260 RETURN
270 :
280 REM ** HIBACSATORNA LEKERDEZESE **
290 :
300 OPEN15,8,15
310 INPUT#15,EN,EM$,ET,ES
320 IF EN<>0 THEN PRINT"HIBAS LEMEZ !":PRINT#15,"I0:":CLOSE15:END
330 RETURN
340 :
350 REM *** REL.POINTER ***
360 :
370 HB=INT(I4/256)
380 LB=I4-HB*256
390 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
400 RETURN
```

A bemutatott program előkészít lemezen egy max. 500 rekordot tartalmazó REL-file-t, amelyben az egyes rekordok hossza max. 31 karakter lehet (240-es sor).

Írás a REL file-ba

Legyen a feladatunk a következő: egy vállalat vásárlásait kell tárolnunk lemezen a következő formában.

A tárolt információ			
jelentése	max. hossza	adattípus	a programváltozó neve
A vásárlás dátuma	8	karakteres	H1\$
A vásárló személy kódja	9	karakteres	H2\$
Az áru egységára	4	numerikus	H3
A vásárolt áru mennyisége	4	numerikus	H6
Az áru kódja	6	karakteres	H7\$

Tehát a nyilvántartás egy rekordja a fenti öt mezőből fog állni. A rekord max. hosszát megkapjuk, ha a mezőhosszakat összeadjuk:

$$RH=8+9+4+4+6=31 \text{ byte,}$$

vagyis egy rekord hasznos hossza 31 byte. Miután a REL-file-ban minden rekordnak azonos hosszúságúnak kell lenni a rövidebb rekordokat ki kell egészítenünk az adott hossza. Ha ezt a megoldást választjuk, akkor az egyes mezőket nem fogjuk tudni egymástól elválasztani.

Ennél jobb megoldás az, ha az aktuális mezőtartalmat a maximális mezőhosszra egészítjük ki, így a rekordon belül mindegyik mező azonos pozíción kezdődik, s a rekordhossz is mindig azonos lesz. Ezt a feladatot végzi el a következő programrészlet.

```
5 REM 93. PROGRAM
50 RC#=H1$+LEFT$(BL$,8-LEN(H1$))
60 RC#=RC#+H2$+LEFT$(BL$,9-LEN(H2$))
70 RC#=RC#+STR$(H3)+LEFT$(BL$,4-LEN(STR$(H3)))
80 RC#=RC#+STR$(H6)+LEFT$(BL$,4-LEN(STR$(H6)))
90 RC#=RC#+H7$+LEFT$(BL$,6-LEN(H7$))
```

Az RC\$-ba rakjuk össze a rekordot, úgy, hogy az egyes mezőkből hiányzó karaktereket szóköz karakterekkel töltjük fel. Így alakítjuk ki az azonos hosszúságú mezőket. A BL\$ tartalma 31 byte-nyi szóköz karakter. Az így elkészített rekordot a következő alprogrammal írhatjuk fel a lemezre:

```
5 REM 94. PROGRAM
10 :
20 REM *** REL.IRO ***
30 :
40 GOSUB 130 :REM POINT.
50 RC#=H1$+LEFT$(BL$,8-LEN(H1$))
60 RC#=RC#+H2$+LEFT$(BL$,9-LEN(H2$))
70 RC#=RC#+STR$(H3)+LEFT$(BL$,4-LEN(STR$(H3)))
80 RC#=RC#+STR$(H6)+LEFT$(BL$,4-LEN(STR$(H6)))
90 RC#=RC#+H7$+LEFT$(BL$,6-LEN(H7$))
100 PRINT#2,RC#
110 RETURN
120 :
130 REM *** REL.POINTER ***
140 :
150 HB=INT(I4/256)
160 LB=I4-HB*256
170 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
180 RETURN
```

Az I4-es változóban adjuk meg azt, hogy hányadik rekordot írjuk fel. Erre pozicionálunk. Az írási műveletet a 100-as sorban végezzük el.

Olvasás a REL-file-ból

Ha az I4-es változóban van az olvasandó rekord kulcsa, akkor erre pozicionálunk. Ezek után egy INPUT utasítással elvégezhetjük az olvasási műveletet. A beolvasott rekordot azonban szét kell szedni a megfelelő mezőkre. Szerencsére ez nem okoz különösebb nehézséget, hiszen az egyes mezők mindig ugyanazon a helyen kezdődnek a rekordon belül.

```
5 REM 95. PROGRAM
10 :
20 REM *** REL. OLVASAS ***
30 :
40 GOSUB 140:REM REL.POINT
50 INPUT#2,AY$
60 IF AY$=CHR$(255) THEN RETURN
70 H1$=MID$(AY$,1,8)
80 H2$=MID$(AY$,9,9)
90 H3=VAL(MID$(AY$,18,4))
100 H6=VAL(MID$(AY$,22,4))
110 H7$=MID$(AY$,26,6)
120 RETURN
130 :
140 REM *** REL.POINTER ***
150 :
160 HB=INT(I4/256)
170 LB=I4-HB*256
180 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
190 RETURN
```

A program az AY\$ változóba olvassa be a teljes rekordot. Ha ennek a tartalma megegyezik a π karakterrel (ASCII kódja 255), akkor az előzőek értelmében egy üres rekordot olvastunk be. Ennek szétszedése értelemszerűen felesleges.

A kialakított REL-file feltöltése — olvasása

Végezetül két teljes programot mutatunk be a REL-file kezelésére, az előzőekben megismert alprogramok, programrészletek felhasználásával.

```
5 REM 96. PROGRAM
10 REM REL.IRO (DEMO)*****
20 :
30 :
40 :
50 N9=8
60 N1$="REL.DATA"
70 GOSUB 190 :REM RELNYIT----->
80 INPUT"REKORDKULCS";I4
90 IF I4=-1 THEN I70
100 INPUT"DATUM" ;H1$
110 INPUT"VASARLO KODJA" ;H2$
120 INPUT"EGYSEGAR" ;H3
130 INPUT"MENNYISEG" ;H6
140 INPUT"ARU KODJA" ;H7$
150 GOSUB 360 :REM IRAS.RELBE----->
160 GOTO80
170 CLOSE2:CLOSE15:END
180 :
190 REM *** REL.FILE MEGNYITASA ***
200 :
210 OPEN2,N9,2,N1$+",".L,"+CHR$(32)
```



```

220 GOSUB 290:REM HIBA
230 BL$=""
240 FOR I=1 TO 32
250 BL$=BL$+" "
260 NEXT I
270 RETURN
280 :
290 REM ** HIBACSATORNA LEKERDEZESE **
300 :
310 OPEN15,8,15
320 INPUT#15,EN,EM$,ET,ES
325 IF EN=0 THEN RETURN
330 IF EN=62 THEN PRINT "A FILE NINCS A LEMEZEN!"
335 PRINT#15,"I0:";CLOSE15:END
340 RETURN
350 :
360 REM *** REL.IRO ***
370 :
380 GOSUB 470:REM POINT.----->
390 RC$=H1$+LEFT$(BL$,8-LEN(H1$))
400 RC$=RC$+H2$+LEFT$(BL$,9-LEN(H2$))
410 RC$=RC$+STR$(H3)+LEFT$(BL$,4-LEN(STR$(H3)))
420 RC$=RC$+STR$(H6)+LEFT$(BL$,4-LEN(STR$(H6)))
430 RC$=RC$+H7$+LEFT$(BL$,6-LEN(H7$))
440 PRINT#2,RC$
450 RETURN
460 :
470 REM *** REL.POINTER ***
480 :
490 HB=INT(I4/256)
500 LB=I4-HB*256
510 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
520 RETURN

```

A program az előzőekben már ismertetett feladat adatait írja lemezre. A rekordok bevitelének végét a 90-es sorszámu utasítással kérdezzük le (I4=-1).

A következő program a már részben felírt REL-file tetszőleges rekordját olvassa be, s képernyőn megjeleníti. Amennyiben az adott kulcsú rekordhoz nem tartozik adat (nem töltöttük még ki), úgy az "ÜRES REKORD" üzenet jelenik meg a képernyőn (105-ös sor). A lekérdezést itt is a 90-es sorszámu utasítás fejezi be (I4=-1).

```

5 REM 97. PROGRAM
10 REM REL.OLVASO(DEMO)*****
20 :
30 :
40 :
50 N3=8
60 N1$="REL.DATA"
70 GOSUB 200:REM RELNYIT----->
80 INPUT"REKORDKULCS=";I4
90 IF I4=-1 THEN N180
100 GOSUB 370:REM RELOLV----->
105 IF AY$="" THEN PRINT "A REKORD URES *****":GOTO170
110 PRINTI4;".REKORD-----"

```

```

120 PRINT"DATUM" "H1$
130 PRINT"VASARLO KODJA "H2$
140 PRINT"EGYSEGAR "H3
150 PRINT"MENNYISEG "H6
160 PRINT"ARU KODJA "H7$
170 GOTO80
180 CLOSE2:CLOSE15:END
190 :
200 REM *** REL.FILE MEGNYITASA ***
210 :
220 OPEN2,N3,2,N1$+".L,"+CHR$(32)
230 GOSUB 300:REM HIBA
240 BL$=""
250 FOR I=1 TO 32
260 BL$=BL$+" "
270 NEXT I
280 RETURN
290 :
300 REM ** HIBACSATORNA LEKERDEZESE **
310 OPEN15,8,15
320 INPUT#15,EN,EM$,ET,ES
325 IF EN=0 THEN RETURN
330 IF EN=62 THEN PRINT "A FILE NINCS A LEMEZEN!"
335 PRINT#15,"I0:";CLOSE15:END
350 RETURN
360 :
370 REM *** REL.OLVASAS ***
380 :
390 GOSUB 490:REM REL.POINT
400 INPUT#2,AY$
410 IF AY$="" THEN RETURN
420 H1$=MID$(AY$,1,8)
430 H2$=MID$(AY$,9,9)
440 H3=VAL(MID$(AY$,18,4))
450 H6=VAL(MID$(AY$,22,4))
460 H7$=MID$(AY$,26,6)
470 RETURN
480 :
490 REM *** REL.POINTER ***
500 :
510 HB=INT(I4/256)
520 LB=I4-HB*256
530 PRINT#15,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
540 RETURN

```


5. FEJEZET

A Disk Monitor

A VC—1541-es lemezmeghajtó egységnek saját mikroprocesszora és operációs rendszere van. A lemez operációs rendszerét DOS-nak hívjuk (Disk Operating System). A DOS-nak nincs szüksége az alapgép tárhelykapacitására, és nem veszi igénybe az alapgép műveleti idejét sem. Az alapgép feladata csak annyi, hogy közvetíti a lemezegységhez a felhasználó utasításait, amelyeket az utóbbi önállóan végrehajt. Az utasítások eredményének ellenőrzése is a DOS feladata, s ha valamilyen hibát érzékel, azt a hibacsatornán keresztül közli az alapgéppel, ill. a felhasználóval.

A következőkben egy olyan hasznos segédprogramot mutatunk be, amely segítségével bepillantást nyerhetünk a lemez szerkezetébe, elolvashatjuk a lemezen tárolt információk területeket, adatterületeket, azokon különböző módosításokat hajthatunk végre. A program neve Disk Monitor.

A 4.1 alfejezetben már szó volt a lemez formátumáról, így a sáv, szektor fogalmak már ismertek.

A Disk Monitor programot a már jól ismert:

```
LOAD"prog-név",8
```

utasítással töltjük be a lemeztől, majd a RUN paranccsal aktivizáljuk. Ekkor a képernyőn a következőket láthatjuk:

```
DISK-MONITOR V1.0
```

```
>
```

ami azt jelenti, hogy a program parancsra vár.

5.1 A DISK MONITOR PROGRAM PARANCSAI

Írjuk be a következő parancsot:

```
>Ru12u00
```

ahol:

R (READ)	egy szektor olvasása lemeztől,
u	a kötelező üres hely jelölése;
12	hexadecimális formában a sáv száma. Ez decimálisan $1 \cdot 16 + 2 = 18$;
00	hexadecimális formában a szektor száma.

A parancsot a RETURN billentyű lenyomásával fejezzük be. Hatására a program beolvassa a lemezről a kívánt szektort, majd újra megjelenik a > jel, jelezvén, hogy a program új parancsra vár.

Az aktuális szektor tartalmát a

>M

paranccsal tudjuk a képernyőn megjelentetni. A szektor képe két fő részből áll. Az első rész hexadecimális számok formájában ábrázolja a lemez tartalmát, a mellette jobbra levő kisebbik rész pedig arra való, hogy ebben a számerdőben könnyen eligazodjunk. Itt láthatóak a hexadecimális értékeknek megfelelő karakterek. A lemeztérkép:

TRACK 18 SECTOR 0

```
00 : 12 01 41 00 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : A  11 11 11
10 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 11 11 11 11
20 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 11 11 11 11
30 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 11 11 11 11
40 : 15 FF FF 1F 0E E0 B7 1F 11 FC FF 07 13 FF FF 07 : 11 11 11 11
50 : 13 FF FF 07 13 FF FF 07 13 FF FF 07 13 FF FF 07 : 11 11 11 11
60 : 13 FF FF 07 12 FF FF 03 12 FF FF 03 12 FF FF 03 : 11 11 11 11
70 : 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01 : 11 11 11 11
80 : 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01 : 11 11 11 11
90 : 4F 4B 54 41 54 41 53 31 A0 A0 A0 A0 A0 A0 A0 : OKTATAS1
A0 : A0 A0 30 31 A0 32 41 A0 A0 A0 A0 A0 A0 A0 : 01 2A
B0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
C0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
D0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
E0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
```

A lemeztérkép részletes ismertetésére a következő fejezetben térünk vissza. Ha egy szektornak csak egy meghatározott részét kívánjuk megjeleníteni, akkor a kívánt rész határait paraméterként beírhatjuk az M utasításba.

A szektor első részének kiírása:

>Mu00u7F

A második rész kiírása:

>Mu80uFF

A beolvasott szektor tartalmát a W paranccsal módosíthatjuk. A kurzormozgató billentyűkkel a hexadecimális területen el kell végezni a javítást. A javítás eredménye a karakteres mezőben a RETURN lenyomása után azonnal látható. Ha a javítást befejeztük, akkor a javított szektort kiírhatjuk a lemezre a következő paranccsal:

>Wu12 00

ahol:

W (WRITE) a Disk-Monitor írási parancsa;
12 hexadecimálisan a sáv száma;
00 hexadecimálisan a szektor száma.

A parancsot itt is a RETURN billentyű lenyomásával fejeztük be. A parancs hatására a program felírja a lemezre a kívánt szektort.

A Disk Monitor program futása közben használható az összes lemezkezelő parancs is. Miután ezek feladatát BASIC-ből már ismerjük, így csak a szintaktikájukat ismertetjük egy rövid megjegyzéssel.

<@ (RETURN) A hibacsatorna leolvasása.

>@I (RETURN) A lemezegység inicializálása (alaphelyzetbe hozása).

>@N:lemeznev,ID A lemez formázása.

>@U: A lemez RESET (bekapcsolási helyzetbe hozása).

>@V: VALIDATE parancs a lemezre.

A Disk Monitor programból a BASIC-be a

>x (RETURN)

paranccsal térhetünk át. Ha a Disk Monitor programot betöltöttük a tárba (s közben a számítógépet egyszer sem kapcsoltuk ki), akkor azt bármikor (betöltés nélkül) újra aktivizálhatjuk a

SYS 49152

paranccsal.

Figyelmeztetés! Mielőtt bárki módosítaná a lemeztartalmat a Disk Monitor programmal, célszerű a lemezt lemásolnia. Különösen akkor, ha még nem tudja biztosan, hogy a beírt értékek mit jelentenek, vagy nincs tisztában a lemez szerkezetével, hiszen a módosítással könnyen végzetes károkat okozhat a saját lemezén!

A következő fejezetben a lemez szerkezetét ismertetjük.

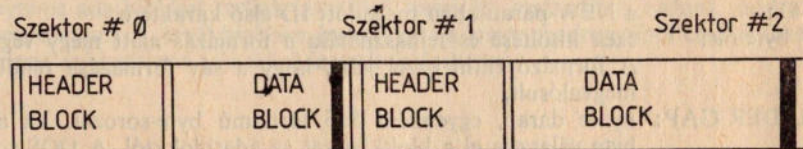
5.2 A SEKTOR FELÉPÍTÉSE

A lemez szerkezetét a DOS a formázás közben alakítja ki (NEW parancs), ekkor hozza létre az 4.1 alfejezetben leírt sáv-, szektorszerkezetet, s elkészít 683 szektort a lemezen. Minden szektor két részből áll.

a) HEADER BLOCK (blokkfej), amely azonosítja a szektort;

b) DATA BLOCK (adatblokk), ez tartalmazza a felhasználói adatokat, a mérete 256 byte.

A 15. ábrán láthatjuk ennek a két résznek a fizikai elhelyezkedését.



szinkronjel
elválasztó köz-gap

15. ábra A szektor felépítése

A szektorok emelkedő számsorrendben alakulnak ki körkörösén, minden sáv esetében. Mint látjuk minden szektor tartalmaz egy azonosító blokkfejezet, amelyet egy adatblokk követ. Az egyes szektorok egymással egy jellel vannak elválasztva, ezt hívják gap-nek (angol szó, ejtsd: gep, jelentése rés, hézag). A másik speciális jel neve szinkronjel, amely jelzi a blokkfej, ill. az adatblokk kezdetét.

A szinkronjel olyab 10 vagy több bitből álló jelsorozat, (alap esetben 40), amely csak az adatblokk vagy a blokkfej kezdetén fordul elő. A lemezegység hardver úton érzékeli ezt a speciális jelsorozatot és jelzi a DOS-nak, hogy egy új adat vagy blokkfej olvasása következik.

A blokkfej szerkezete

Mint tudjuk, a blokkfej az adott szektor azonosítására való. Tartalma egy szinkronjel, nyolc információs byte és egy gap. A szerkezetét a 16. ábra szemlélteti.

Szinkron- jel	Blokk- fej ID	Kontroll- összeg	Szektor szám	Sáv szám	ID #2	ID #1	S0F byte	S0F byte	Header GAP
------------------	------------------	---------------------	-----------------	-------------	----------	----------	-------------	-------------	---------------

16. ábra A blokkfej szerkezete

Nézzük meg az egyes részek jelentését!

- Szinkronjel:** tíz vagy több bitből álló jelsorozat. Jelzi a DOS számára, hogy blokkfej olvasása következik és nem adatblokké.
- Blokkfej ID:** alap esetben tartalma \$08, ez is jelzi a DOS számára, hogy amit olvas, az a blokkfej.
- Kontrollösszeg:** egy ellenőrző szám, amelynek értékét a sáv- és a szektorszám, továbbá a két ID karakterből képzik az operációs rendszer egy kizáró logikai VAGY művelettel (EOR). Ennek a számnak a képzését minden olvasáskor elvégzi a DOS és a kapott eredményt összehasonlítja a tárolt értékkel. Eltérés esetén az olvasásban hiba lépett fel. A kontroll összegek a lemez formázásakor íród-
nak be.
- Szektorszám:** a kérdéses szektor számát tartalmazza. Értékét a DOS formá-
záskor írja be.
- Sávszám:** a kérdéses sáv száma. Szintén formázás közben íródik be. A szektorszámmal együtt a DOS ellenőrzi, hogy a fejbeállítás (pozicionálás) helyes volt-e.
- ID#2:** a NEW parancsban megadott azonosító (ID) második karak-
tere. A DOS a lemezcsereit ebből is észreveheti.
- ID #1:** a NEW parancsban megadott ID első karaktere.
- \$0F byte-ok:** ezek kitöltése és felhasználása a formázás alatt megy végbe. A formázó rutin ezzel jelzi, hogy a sáv formázása rendben megvalósult.
- HEADER GAP:** nyolc darab, egyenként \$55 tartalmú byte-sorozat. Ez a 8 byte választja el a blokkfejezet az adatbloktól. A DOS soha sem olvassa be ezeket a byte-okat, amelyek arra való, hogy legyen ideje a DOS-nak felkészülni az adatblokk olvasására. Megjegyzés: a 4040-es meghajtó kilenc gap byte-ot tartalmaz, s ez az egyik oka annak, hogy a 1541-es meghajtó nem írás-
kompatibilis a 4040-es egységgel.

Az adatblokk szerkezete

Minden adatblokk 256 byte-nyi adatot tud tárolni. Tartalmaz továbbá: szinkronjelet, adatblokk-azonosító karaktert (ID), adatblokk ellenőrző összeget két \$00 byte-ot. Ezek elhelyezkedését az adatblokkon belül a 17. ábra szemlélteti.

Szinkron- jel	Adatblokk ID	256 byte adat	Kontroll- összeg	\$00 byte	\$00 byte	IS GAP
------------------	-----------------	------------------	---------------------	--------------	--------------	-----------

17. ábra Az adatblokk szerkezete

Nézzük meg az egyes részek jelentését!

- Szinkronjel:** funkcióját tekintve ua., mint azt az első részben leírtuk.
- Adatblokk ID:** alap esetben tartalma \$07. Arra való, hogy jelezze a DOS szá-
mára, hogy amit olvas, az adatblokk és nem blokkfej (\$08). az aktuálisan tárolt adatokat tartalmazza, az első két byte kivé-
telével. Ezt ui. az adatblokkok logikai egymáshoz láncolásához használja, sáv/szektor mutatóként (pointerként).
- 256 byte adat:** ellenőrző összeg a DOS számára. Ha az aktuálisan képzett, s a
tárolt ellenőrzőszám nem egyezik, akkor az olvasást a DOS hibásnak ítéli meg, s újra próbál olvasni. Az ellenőrző számot a DOS a 256 byte között végzett EOR művelettel képz.
- Kongrollösszeg:** ua. mint a blokkfejnél leírt \$0F byte
- \$00 byte:** az adatblokk és a következő szektor kezdete között helyezkedik
el. A hossza változó és különbözik lemezmeghajtónként is (azo-
nos típuson belül is!). Jól működő egységek esetében a hossza 4—12 byte lehet. (Az utolsó szektor után viszont a gap hossza
gyakran 100 byte is lehet.) Fontos, hogy ejég hosszú legyen, mert egy adatblokk írásánál a lemezegységen fellépő fordulat-
szám-ingadozások miatt könnyen felülírhatjuk a következő szek-
tor kezdetét. (A lemezmeghajtó általában 300 1/min fordulat-
számmal forgatja a behelyezett lemezt. Ez a fordulatszám függ-
vénye a hálózati feszültség ingadozásának, a mechanikai szer-
kezetek állagának és a külső hőmérséklet ingadozásának).

Az eddigiekből látható, hogy a lemezen nagyon sok olyan információ található, amit a lemezegység hardvere használ fel. A Disk Monitor programmal csak a 256 byte-os adatblokkot tudjuk láthatóvá tenni, ill. csak abban tudunk módosításokat végrehajtani. A továbbiakban éppen ezért a lemez szerkezetéből csak a Disk Moni-
rorral módosítható részekkel fogunk foglalkozni.

5.3 A LEMEZ LOGIKAI FELÉPÍTÉSE

5.3.1 A VC—1541-es BAM-ja

A BAM (Block Availability Map, (a blokkok foglaltsági térképe) feladata, hogy a szektorok (blokkok) foglaltságát nyilvántartsa. Tehát ez egy lemeztárképnak is tekinthető. A Bam a 18-as sáv 0-ás szektorában található, a DOS a lemez formázása közben alakítja ki.

Nézzük meg a BAM szerkezetét!

TRACK 18 SECTOR 0

```
00 : 12 01 41 00 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : A 44 44 44
10 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 44 44 44 44
20 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 44 44 44 44
30 : 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : 44 44 44 44
40 : 15 FF FF 1F 0E E0 B7 1F 11 FC FF 07 13 FF FF 07 : 44 44 44 44
50 : 13 FF FF 07 13 FF FF 07 13 FF FF 07 13 FF FF 07 : 44 44 44 44
60 : 13 FF FF 07 12 FF FF 03 12 FF FF 03 12 FF FF 03 : 44 44 44 44
70 : 12 FF FF 03 12 FF FF 03 12 FF FF 03 11 FF FF 01 : 44 44 44 44
80 : 11 FF FF 01 11 FF FF 01 11 FF FF 01 11 FF FF 01 : 44 44 44 44
90 : 4F 4B 54 41 54 41 53 31 A0 A0 A0 A0 A0 A0 A0 : OKTATAS1
A0 : A0 A0 30 31 A0 32 41 A0 A0 A0 A0 00 00 00 00 : 01 2A
B0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
C0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
D0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
E0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
```

Mint minden szektor, a BAM is 256 byte-ból áll. A kettőspont előtti hexadecimális szám értéke mutatja az adott sorban látható byte-ok közül az első sorszámot. A következő táblázat a BAM szerkezetét mutatja.

A byte sorszáma (dec.)	A byte sorszáma (hex.)	Tartalom (hex.)	Megjegyzés
0/1	\$0/\$1	\$12/\$1	A tartalomjegyzék kezdetét jelzi (sáv/ szektor pointer)
2	\$2	\$41	Az A karakter ASCII kódja, s jelzi a DOS számára, hogy a lemezt 1541-es egységen formáztuk meg.
3	\$3	\$0	Nem használatos.
4—143	\$4—\$8F	—	Blokkok foglaltsági térképe (BAM)
144—159	\$90—\$9F	—	A lemez neve SHIFT-elt szóközjellel kiegészítve.
160—161	\$A0—\$A1	\$A0	SHIFT-elt szóközjel.
162—163	\$A2—\$A3	—	A lemez ID karaktere.
164	\$A4	\$A0	SHIFT-elt szóközjel.
165—166	\$A5—\$A6	\$32/\$41	A DOS változat formát típusa (2A).
167—170	\$A7—\$AA	\$A0	SHIFT-elt szóközjel.
170—255	\$AB—\$FF	?	Nem használatos.

A DOS négy byte-ot használ fel minden sáv leírására. Mivel 35 sávunk van, így $4 \times 35 = 140$ byte szükséges a BAM kialakításához (\$04-től \$8F-ig terjedő rész).

Vizsgáljuk meg ezt a négy byte-ot a 18-as sáv adatain keresztül! Ezek a BAM-ban a

$18 \times 4 = 72$ (\$48)

címen kezdődnek, s tartalmuk:

\$40:.....11 FC FF 07.....

Az első byte tartalma \$11 jelzi a 18-as sávban még szabadon levő szektorok számát. Általában minden négyes byte-csoport első tagja a kérdéses sávban levő szabad szektorok számát adja. Tehát a 18-as sávban

$\$11 = 1 \times 16 + 1 = 17$ szektor szabad.

Egy kis számolással mi is meghatározhatjuk, hogy a lemeziünkön mennyi szabad szektor (blokk) van.

Sorszám	Hex. érték	Dec. érték
1—16	\$15	21
17	\$0E	14
18	\$11	17
19—24	\$13	19
25—30	\$12	18
31—35	\$11	17

Vagyis

$16 \times 21 + 14 + 17 + 6 \times 19 + 6 \times 18 + 5 \times 17 = 674$

Tehát a lemeziünkön 674 szektor szabad. Nézzük meg, hogy a lemez tartalomjegyzékében mit láthatunk!

```
0 OKTATAS1 01 2A
1 "PROGRAM1" PRG<
2 "SZAMOK" SEQ
1 "PROGRAM2" PRG
1 "SZOVEG" SEQ
2 "DEC->BIN" PRG
657 BLOCKS FREE.
```

READY.

A tartalomjegyzék szerint 657 szektor szabad a lemezen. Elvettük volna a számolást? Erről szó sincs! A DOS a 18-as sávot teljes mértékben saját céljaira használja, így az számunkra nem szabad terület. Az összegzésben ezt természetesen figyelembe vettük (l. az előző táblázatot). Mivel a 18-as sávban 17 szektor még szabad, úgy ezt a kapott értékből levonva:

szabad szektort kapunk. (Eljárhatunk úgy is, hogy a 18-as sáv szabad szektorait nem adjuk hozzá a felhasználói szabad szektorokhoz, s így a helyes eredményt azonnal megkapjuk.)

Térjünk vissza a maradék három byte jelentésére (*).

```
$40:.....11 FC FF 07
      * * *
Hex.:      $FC      $FF      $07
Dec.:      252      255      7
Bin:      11111100  11111111  00000111
              11111111  21111
Szektor:   76543210  54321098  XXX09876
```

Megjegyzés: 1 = szabad blokk
0 = foglalt blokk

Tehát az első byte, amelynek tartalma \$FC, leírja a 0-ástól a 7-esig terjedő szektorok foglaltságát. Esetünkben a 0-ás és az 1-es szektorok foglaltak (0 bit), a többi szabad (1 bit).

A következő byte, amelynek tartalma \$FF, a 8-astól a 15-ösig terjedő szektorok foglaltságát ábrázolja. Esetünkben ezek mind szabad szektorok (végig 1 bit).

A harmadik byte, amelynek tartalma \$07, leírja a 16-ostól a 20-asig terjedő szektorok foglaltságát.

A többi bit jelentése értelmetlen, hiszen a lemeztérkép szerint (l. a 4.1 alfejezetben) egy sávon maximálisan 21 szektor lehet (0-20). A 18-as sávon viszont 19 szektor van (0-18), tehát számunkra csak ezek a bitek értelmezhetők, a tartalmuk alapján a 16-os-tól a 18-asig terjedő szektorok is szabadok.

A bemutatott számítási eljárások megkönnyítésére egy konverziós programot közlünk, amely a beírt szám bináris alakját adja vissza. A decimálisból hexadecimálisba, ill. fordított konverziót célszerű a HELP+segédprogrammal elvégezni (l. a Függelékben).

```
100 REM DECIMALISBOL BINARISRA KONVERZIO
110 :
120 DEF FN B(B)=2*(B-INT(B/8)*8) AND 0
130 PRINT "BOL DECIMALISBOL BINARISRA"
140 D=-1
150 INPUT "DECIMALIS SZAM:";D
160 IF D<0 OR D>255 THEN END
170 PRINT "BINARIS SZAM:";
180 FOR B=7 TO 0 STEP -1
190 IF FNB(B)=0 THEN PRINT "0";:GOTO 210
200 PRINT "1";
210 NEXT B
220 PRINT
230 GOTO 140
```

READY.

5.3.2 A tartalomjegyzék szerkezete

Emlékezzünk arra, hogy a 18-as sávon a felhasználó nem tárolhat adatokat. Ez végig az operációs rendszer számára van fenntartva. Azt már láttuk az előző fejezetben, hogy a 18-as sáv 0-ás szektorában a BAM található. De vajon mit tartalmaz a többi szektor?

Nos, ezekben található a lemez tartalomjegyzéke, bár nem egészen olyan formában, mint azt korábban megszoktuk. Olvassuk be a lemez tartalomjegyzékét és írasuk ki!

LOAD"\$",8

LIST

Pl.: a következőket láthatjuk:

```
0  "PROGRAM1"  PRG
1  "SZAMOK"    SEQ
1  "PROGRAM2"  PRG
1  "SZOVEG"    SEQ
2  "DEC->BIN"  PRG
657 BLOCKS FREE.
```

READY.

Ennek a szerkezetét már ismerjük. Most nézzük meg a tartalomjegyzéket úgy, ahogyan az a lemezen található. Mint tudjuk (a BAM szerkezetének leírásából), a lemez tartalomjegyzéke a 18-as sáv 1-es szektorában kezdődik. Olvassuk be ezt a Disk Monitor programmal:

>R_12_01
>M

A tartalomjegyzék belső felépítését látjuk:

TRACK 18 SECTOR 1

```
00 :00 FF 02 11 00 50 52 4F 47 52 41 40 31 A0 A0 A0 : ■ PROGRAM1
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 01 00 :
20 :00 00 01 11 01 53 5A 41 40 4F 4B A0 A0 A0 : ■ SZAMOK
30 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 02 00 :
40 :00 00 02 11 02 50 52 4F 47 52 41 40 32 A0 A0 : ■ PROGRAM2
50 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 01 00 :
60 :00 00 01 11 03 53 5A 4F 56 45 47 A0 A0 A0 : ■ SZOVEG
70 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 01 00 :
80 :00 00 02 11 04 44 45 43 2D 3E 42 43 4E A0 A0 : ■ DEC->BIN
90 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 02 00 :
A0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
B0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
C0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
D0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
E0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
F0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
```

A következő táblázatban összefoglaltuk a tartalomjegyzék szektorok szerkezetét.

Byte (hex.)	Tartalom	Megjegyzés
\$0	—	A következő sáv mutatója a tartalomjegyzékben.
\$01	—	A következő szektor mutatója a tartalomjegyzékben.
\$2—\$1E	—	Az 1. file bejegyzése a tartalomjegyzékben.
\$20—\$21	0	Nem használt.
\$22—\$3F	—	A 2. file bejegyzése
\$40—\$41	0	Nem használt.
\$42—\$5F	—	A 3. file bejegyzése
\$60—\$61	0	Nem használt.
\$62—\$7F	—	A 4. file bejegyzése
\$80—\$81	0	Nem használt.
\$82—\$9F	—	Az 5. file bejegyzése
\$A0—\$A1	0	Nem használt.
\$A2—\$BF	—	A 6. file bejegyzése
\$C0—\$C1	0	Nem használt.
\$C2—\$DF	—	A 7. file bejegyzése
\$E0—\$E1	0	Nem használt.
\$E2—\$FF	—	A 8. file bejegyzése

Egy tartalomjegyzék szektorban 8 file-bejegyzés fér el, s mivel 18 szektorunk van, összesen:

$$8 * 18 = 144$$

file-bejegyzés lehet egy lemezen. Minden szektor első két byte-ja egy mutató, amely a logikailag következő sáv és szektor számát tartalmazza. A mi példánkban ezek értéke 00 FF, ami azt jelenti, hogy a tartalomjegyzék nem folytatódik más szektorokban, ez az utolsó szektora. Abban az esetben, ha a tartalomjegyzékben 8-nál több bejegyzés lenne, akkor a láncolási mutató 12 04 lenne. Azt, hogy melyik lesz a következő szektor, a tartalomjegyzék esetében, a következő kis táblázatból előre megállapíthatjuk.

0 BAM

1, 4, 7, 10, 13, 16

2, 5, 8, 11, 14, 17

3, 6, 9, 12, 15, 18

(a 18-as sáv szektorainak logikai lánc)

Tehát az 1-es szektort nem a 2-es szektor követi logikailag, hanem a 4-es, azután pedig a 7-es stb. Miért van ez így? Emlékezzünk arra, hogy a lemezegységünk 300 l/min fordulatszámmal forgatja a behelyezett lemezt. Míg a beolvasott adatok továbbítását a lemezegység hardvere, szoftvere elvégzi, s felkészül a következő szektor olvasására, addig a mechanika a lemezt kb. 2 szektorral elforgatja. Azért, hogy ne kelljen sokat várakozni arra, amíg az olvasandó szektor újra az olvasófej pozíciójához ér, bevezették az előbbi láncolási módszert. Amikor a lemezegység végzett egy szektor továbbításával, az olvasófej előtt pontosan a következő olvasandó szektor áll, tehát a várakozási idő minimális (hasonlóan magyarázható írás esetén is).

A következőkben vizsgáljunk meg egy file-bejegyzést a tartalomjegyzékben!

TRACK 18 SECTOR 1

```
00 :00 FF 02 11 00 50 52 4F 47 52 41 40 31 A0 A0 A0 : ■ PROGRAM1
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 01 00 :
```

Byte (hex.)	Megjegyzés
\$2	A file-típus kódja.
\$3—\$4	A file fizikai helyének kezdete a lemezen (sáv/szektor pointer).
\$5—\$14	A file neve (A0 karakterekkel 16-ig feltöltve).
\$15—\$17	REL-file-ok használják.
\$18—\$1B	Nem használatos.
\$1C—\$1D	A @; prog. név alakú file-ok felülírásánál használatos.
\$1E—\$1F	A file hossza szektorokban kifejezve (alsó byte, felső byte tárolási formában).

A file-bejegyzés kódja alapján a DOS eldönti a file típusát, s azt a megszokott formában a tartalomjegyzékben megjeleníti.

A következő file-típusok, ill. kódok a megengedettek:

Kód	Jelentés
\$00	Törölt file. A tartalomjegyzékben nem látható.
\$80	A file törölt (DEL)
\$81	Soros file (SEQ)
\$82	Programfile (PRG)
\$83	User file (USR)
\$84	Relatív file (REL)

Példánkban a PROGRAM 1 file-bejegyzés kódja \$82, tehát ez egy programfile. A táblázatból láthatjuk, hogy minden file-kód nyolccsal kezdődik. A kezdőérték alapján még a következő kódértelmezéseket tudjuk tenni: a \$0-val kezdődő kódok, a nem lezárt file-ra utalnak. A tartalomjegyzékben ezeket a file-típus előtti * karakter jelzi. Tehát:

\$01 *SEQ
\$02 *PRG
\$03 *URS
\$04 a nem lezárt REL-file, de ez nem fordulhat elő

Ha egy file-t felülírunk, akkor a neve előtt egy @ : karaktert jelzünk. A felírás folyamata alatt a DOS ezt úgy jegyzi meg, hogy a file-típus kódját \$A-val kezdi. Értelemszerűen a \$A4-es kód nem fordulhat elő, mert a REL-file felülírás esetén fizikailag ugyanott marad a lemezen.

Nagyon hasznos és sokszor használjuk is a \$C-vel kezdődő file-típus kódokat. Az ilyen kódú file a törlés (SCRATCH) ellen védett.

A tartalomjegyzékben a védett file-okat a típusjelzés után egy < jel jelöli.

\$C0 DEL<
\$C1 SEQ<
\$C2 PRG<
\$C3 USR<
\$C4 REL<

Gyakorlasképpen alkalmazzuk az utóbbi kódot a PROGRAM 1 nevű program védelmére. A tartalomjegyzékben a módosítást a Disk Monitor programmal végezzük el:

>R_12_01

Beolvassuk a tartalomjegyzék első szektorát:

>M_00_20

Kiíratjuk képernyőre a PROGRAM 1 nevű file bejegyzéseit.

TRACK 18 SECTOR 1

```
00 :00 FF 82 11 00 50 52 4F 47 52 41 4D 31 A0 A0 A0 : 11 PROGRAM1
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
20 :00 00 81 11 01 53 5A 41 4D 4F 4B A0 A0 A0 A0 : 12 SZAMOK
```

A kurzormozgató billentyűkkel a harmadik byte-on levő \$82 hex. értéket átírjuk \$C2-re, majd megnyomjuk a RETURN billentyűt. A következőket fogjuk látni:

TRACK 18 SECTOR 1

```
00 :00 FF C2 11 00 50 52 4F 47 52 41 4D 31 A0 A0 A0 : 11 PROGRAM1
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
```

Írjuk vissza a lemezre a 18-as sáv 1-es szektorát:

>W_12_01

Ezzel programunkat elláttuk törlésvédelemmel.
Lépünk ki a Disk Monitor programból:

>X

és olvassuk be a tartalomjegyzéket:

```
0 SCRATCHES 1 01 20
1 "PROGRAM1" PRG
2 "SZAMOK" SEQ
1 "PROGRAM2" PRG
1 "SZOVEG" SEQ
2 "DEC->BIN" PRG
657 BLOCKS FREE.
```

READY. ^

Listázás után a file-kód mögött egy < jelet látunk. Próbáljuk meg törölni a programot a SCRATCH paranccsal! Ha valaki a próbálkozás hevében véletlenül a PROGRAM 2 nevű programot törölné, akkor sem kell elkeserednie. A törlési parancs hatá-sára a DOS ui. valójában nem a file-t törli, hanem a file-bejegyzést módosítja.


```

00 : 00 FF 82 11 00 50 52 4F 47 52 41 4D 31 A0 A0 A0 :  PROGRAM1
10 : A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
20 : 00 00 81 11 01 53 5A 41 4D 4F 4B A0 A0 A0 A0 :  SZAMOK
30 : A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 02 00 :
40 : 00 00 00 11 02 50 52 4F 47 52 41 4D 32 A0 A0 : PROGRAM2
50 : A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
60 : 00 00 81 11 03 53 5A 4F 56 45 47 A0 A0 A0 A0 :  SZOVEG
70 : A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
80 : 00 00 82 11 04 44 45 43 2D 3E 42 43 4E A0 A0 A0 :  DEC->BIN
90 : A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 02 00 :
A0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
B0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
C0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
D0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
E0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :

```

A \$43 byte tartalma \$00, s ez a kód jelzi a DOS számára, hogy ez a bejegyzés törölt. Ha a törlés után nem írtunk semmit sem a lemezre, akkor a törölt programot maradéktalanul visszakaphatjuk. Miután tudjuk, hogy ez a bejegyzés egy programfile volt, s ennek a kódja \$82, ezt visszaírva a lemezre, a program ismét elérhetővé válik. Ne feledkezzünk meg azonban egy fontos tényről! A bejegyzés törlésekor nemcsak a tartalomjegyzék módosul, hanem az általa elfoglalt szektorok is felszabadulnak, amit a BAM-ban ellenőrizhetünk.

A visszaállított program által elfoglalt szektorokat újra le kell foglalni, ami egyenértékű a BAM módosításával. Ezt legegyszerűbben a VALIDATE paranccsal végezhetjük el.

 $\geq @V:$

A parancs hatására a DOS végigmegy a teljes tartalomjegyzéken, a bejegyzések áncolatán, s a BAM-ban újra lefoglalja az aktív területeket.

5.4 A PROGRAMFILE-OK TÁROLÁSA LEMEZEN

Vizsgáljuk meg, hogy a DOS hogyan tárolja a PROGRAM1 nevű file-t a lemezen!

A tartalomjegyzékből tudjuk, hogy a program fizikailag a \$11-es sáv \$00-ás szektorán helyezkedik el. A programfile felépítése:

TRACK 18 SECTOR 1

```
00 :00 FF C2 11 00 50 52 4F 47 52 41 4D 31 A0 A0 A0 : #1 PROGRAM1
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 01 00 :
```

Ezt olvashatjuk ki a \$3 és \$4 byte-ból.

Olvassuk be a Disk Monitor programmal a \$11-es sáv \$00-ás szektorát (a 71-es sáv 0-ás szektora decimálisan).

```

00 :00 45 01 08 19 08 0A 00 3F 31 2C 38 2C 32 2C 22 : E      1,8,2,"
10 :53 5A 41 4D 4F 4C 2D 53 2C 57 22 00 29 08 14 00 :SZAMOK,S,W" )
20 :81 20 49 B2 31 20 A4 20 31 30 30 32 08 1E 00 :I I+1 - 100 2
30 :98 31 2C 49 00 38 08 28 00 82 00 3F 08 32 00 A0 :I, I 8 ( 0 ? 2
40 :31 00 00 00 BC 22 AD 38 29 AC 38 29 20 AF 20 44 :1      "19) 3) _ D
50 :00 71 08 82 00 99 22 93 11 11 20 44 45 43 43 :  国屋"读屋  DECI
60 :4D 41 4C 49 53 42 4F 4C 20 42 49 4E 41 52 49 53 :MALISIBO BINARIS
70 :52 41 22 00 7A 08 8C 00 44 B2 AB 31 00 95 08 96 :RA"  读屋 D+1 读屋
80 :00 85 22 11 11 20 44 45 43 49 4D 41 4C 49 53 20 :  读屋  DECIMALIS
90 :53 5A 41 4D 22 3B 44 00 AB 08 A0 00 3B 20 44 B3 :SZAM"1D 1 读屋 D1
A0 :30 20 B0 20 44 B1 32 35 35 20 A7 20 80 00 C4 08 :0 1 D+255 1 读屋
B0 :AA 00 99 22 11 11 20 42 49 4E 41 52 49 53 20 53 :1 读屋  BINARIS S
C0 :5A 41 4D 3A 22 3B 00 D7 08 B4 00 81 20 42 B2 37 :ZAM:" 0 1 读屋 B+7
D0 :20 A4 20 30 20 A9 20 AB 31 00 F3 08 BE 00 8B 20 :_0 1 1 1 读屋
E0 :A5 42 28 42 29 B2 30 20 A7 20 99 22 30 22 3B 3A :1 B(B)+0 1 读屋"0"
F0 :83 20 32 31 30 00 FD 08 C8 00 99 22 31 22 3B 00 :B210 1 读屋"1"

```

A láncolás első látásra kissé bonyolultnak tűnik, azonban a szerkezete nagyon egyszerű (18. ábra).

A köv. sáv száma	A köv. szektor száma	Betölt. cím alsó (LB)	Betölt. cím felső (HB)	252 adatblokk
------------------	----------------------	-----------------------	------------------------	---------------

18. ábra Láncolás

Az első két byte, a láncolási cím, mint azt a tartalomjegyzéknél már láttuk jelen esetben a program csak egy blokk hosszú, így a következő sáv száma \$00-t tartalmaz. A következő szektor száma pedig azt mutatja, hogy a blokkon belül hány byte tartozik a programhoz. Esetünkben az \$43. A szektorban levő többi adat az előző lemezműveletek során ott maradt „szemét”.

A következők két byte azt jelenti, hogy az alapgép RAM-jának melyik címétől kell elkezdenni a program betöltését. Röviden ezt a program betöltési címének nevezzük. A cím egy alsó byte és egy felső byte formátumban tárolódik. Példánkban az alsó byte (LB) \$01, a felső byte (HB) \$08. Vagyis a tényleges betöltési cím:

HBLB formájú, tehát \$0801

ami megfelel 2049 decimális címnek. Minden BASIC program a tárban ezen a címen kezdődik.

Emlékezzünk vissza a LOAD parancsra! Ha így adjuk ki:

LOAD"prog.név",8

akkor a lemezen tárolt betöltési címet a DOS átlépi és automatikusan a \$0801-es címtől (2049) kezdve tölti be a programot a tárba (függetlenül attól, hogy a lemezen milyen betöltési cím van). Ezt csak akkor használhatjuk, ha biztosak vagyunk abban, hogy a betöltendő program BASIC program!

Ha a:

LOAD"pr.név",8,1

formában adjuk ki a parancsot, akkor minden esetben a lemezen tárolt címtől kezdve végzi el programbetöltést.

Ha tehát nem tudunk semmit a betöltendő programról, akkor célszerű az utóbbi formulát használni, hiszen ha BASIC programról van szó, az mindig a megfelelő helyre fog betöltődni. Ha azonban a program gépkódú, akkor feltétlenül a „helyére” kell tölteni, mert különben nem fog működni (l. pl. HELP+).

Természetesen, ha a program több szektort foglal el, akkor a következő szektorok a betöltési címet már nem tartalmazzák (azt csak az első szektor tartalmazza). A szektor szerkezete a 19. ábra szerintre módosul:

A köv. sáv száma	A köv. szektor száma	254 byte adat
------------------	----------------------	---------------

19. ábra Programfile láncolása

5.5 A SOROS FILE TÁROLÁSA LEMEZEN

Írjuk be a következő programot, majd futtassuk le!

```
10 OPEN1,8,2,"SZAMOK,S,W"  
20 FOR I=1 TO 100  
30 PRINT#1,I  
40 NEXT  
50 CLOSE1
```

READY.

A program létrehoz lemezen egy SZAMOK nevű soros állományt, s 1-től 100-ig kiírja az adatokat.

A tartalomjegyzékben a következő bejegyzést látjuk:

TRACK 18 SECTOR 1

```
00 :00 FF C2 11 00 50 52 4F 47 52 41 4D 31 A0 A0 A0 : 41 PROGRAM1  
10 :A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 01 00 :  
20 :00 00 81 11 01 53 5A 41 4D 4F 4B A0 A0 A0 A0 : 23 SZAMOK  
30 :A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 02 00 :
```

Tehát a file a \$11-es sáv \$01-es szektorán kezdődik. Írassuk ki:

TRACK 17 SECTOR 1

```
00 :11 0B 20 31 20 0D 20 32 20 0D 20 33 20 0D 20 34 : 1 2 3 4  
10 :20 0D 20 35 20 0D 20 36 20 0D 20 37 20 0D 20 38 : 5 6 7 8  
20 :20 0D 20 39 20 0D 20 31 30 20 0D 20 31 31 20 0D : 9 10 11  
30 :20 31 32 20 0D 20 31 33 20 0D 20 31 34 20 0D 20 : 12 13 14  
40 :31 35 20 0D 20 31 36 20 0D 20 31 37 20 0D 20 31 : 15 16 17 1  
50 :38 20 0D 20 31 39 20 0D 20 32 30 20 0D 20 32 31 : 18 19 20 21  
60 :20 0D 20 32 32 20 0D 20 32 33 20 0D 20 32 34 20 : 22 23 24  
70 :0D 20 32 35 20 0D 20 32 36 20 0D 20 32 37 20 0D : 25 26 27  
80 :20 32 38 20 0D 20 32 39 20 0D 20 33 30 20 0D 20 : 28 29 30  
90 :33 31 20 0D 20 33 32 20 0D 20 33 33 20 0D 20 33 : 31 32 33 3  
A0 :34 20 0D 20 33 35 20 0D 20 33 36 20 0D 20 33 37 : 34 35 36 37  
B0 :20 0D 20 33 38 20 0D 20 33 39 20 0D 20 34 30 20 : 38 39 40  
C0 :0D 20 34 31 20 0D 20 34 32 20 0D 20 34 33 20 0D : 41 42 43  
D0 :20 34 34 20 0D 20 34 35 20 0D 20 34 36 20 0D 20 : 44 45 46  
E0 :34 37 20 0D 20 34 38 20 0D 20 34 39 20 0D 20 35 : 47 48 49 5  
F0 :30 20 0D 20 35 31 20 0D 20 35 32 20 0D 20 35 33 : 50 51 52 53
```

Mivel soros adatállományról van szó, a betöltési cím itt értelmetlen, nem is szerepel. A szektor szerkezete a 20. ábrán látható.

A köv. sáv száma	A köv. szektor száma	254 adat byte
------------------	----------------------	---------------

20. ábra Soros file láncolása

Látható, hogy a következő adatblokk a lemezen a \$11-es sáv \$0B szektorában található. Ez a következőképpen néz ki:

TRACK 17 SECTOR 11

```
00 :00 F1 20 0D 20 35 34 20 0D 20 35 35 20 0D 20 35 : 54 55 5  
10 :36 20 0D 20 35 37 20 0D 20 35 38 20 0D 20 35 39 : 56 57 58 59  
20 :20 0D 20 36 30 20 0D 20 36 31 20 0D 20 36 32 20 : 60 61 62  
30 :0D 20 36 33 20 0D 20 36 34 20 0D 20 36 35 20 0D : 63 64 65  
40 :20 36 36 20 0D 20 36 37 20 0D 20 36 38 20 0D 20 : 66 67 68  
50 :36 39 20 0D 20 37 30 20 0D 20 37 31 20 0D 20 37 : 69 70 71 7  
60 :32 20 0D 20 37 33 20 0D 20 37 34 20 0D 20 37 35 : 72 73 74 75  
70 :20 0D 20 37 36 20 0D 20 37 37 20 0D 20 37 38 20 : 76 77 78  
80 :0D 20 37 39 20 0D 20 38 30 20 0D 20 38 31 20 0D : 79 80 81  
90 :20 38 32 20 0D 20 38 33 20 0D 20 38 34 20 0D 20 : 82 83 84  
A0 :38 35 20 0D 20 38 36 20 0D 20 38 37 20 0D 20 38 : 85 86 87 8  
B0 :38 20 0D 20 38 39 20 0D 20 39 30 20 0D 20 39 31 : 89 90 91  
C0 :20 0D 20 39 32 20 0D 20 39 33 20 0D 20 39 34 20 : 92 93 94  
D0 :0D 20 39 35 20 0D 20 39 36 20 0D 20 39 37 20 0D : 95 96 97  
E0 :20 39 38 20 0D 20 39 39 20 0D 20 31 30 30 20 0D : 98 99 100  
F0 :04 FA 00 00 00 00 00 00 00 00 00 00 00 00 : J
```

A karakteres mezőben láthatjuk a felírt számokat. Ha jól megnézzük a hexadecimális táblázatot, akkor bizonyos szabályszerűségeket fedezhetünk fel az adatok tárolásában. Pl. a kettes szám tárolása a lemezen a következő:

0D 20 32 20 0D

ahol:

0D a RETURN kódja (13) adatok elválasztására;
32 a 2-es szám ASCII-kódja (hex.);
20 a szóköz ASCII-kódja (hex.).

Minden szám tárolásának módja ehhez hasonló (21. ábra).

Szóköz	Számjegy	Szóköz	RETURN
1. kód	1. kód	1. kód	1. kód

21. ábra Számok tárolása

Minden szám annyi byte-ot foglal el a lemezen, amennyi számjegyből áll. Ha ehhez még hozzáadunk 3 byte-ot (két szóköz és egy RETURN), könnyen meg tudjuk becsülni az adatállomány lemezhelyigényét.

Nézzük meg, hogy a DOS hogyan tárolja a szövegeket! Ehhez írjuk be a következő segédprogramot:

```
10 OPEN1,8,2,"SZOVEG,S,W"  
20 FOR I=1 TO 10  
30 PRINT#1,"PROBAFELIRAT"  
40 NEXT  
50 CLOSE1
```

READY.

A program a lemezen a SZOVEG nevű file-ba felírja tízszer egymás után a

"PROBAFELIRAT"

szöveget.

Az adatállomány listája a Disk Monitorral:

TRACK 17 SECTOR 3

```
00 :00 85 50 52 4F 42 41 46 45 4C 49 52 41 54 0D 50 : ■PROBAFELIRAT P  
10 :52 4F 42 41 46 45 4C 49 52 41 54 0D 50 52 4F 42 :ROBAFELIRAT PROB  
20 :41 46 45 4C 49 52 41 54 0D 50 52 4F 42 41 46 45 :AFELIRAT PROBAFE  
30 :4C 49 52 41 54 0D 50 52 4F 42 41 46 45 4C 49 52 :LIRAT PROBAFELIR  
40 :41 54 0D 50 52 4F 42 41 46 45 4C 49 52 41 54 0D :AT PROBAFELIRAT  
50 :50 52 4F 42 41 46 45 4C 49 52 41 54 0D 50 52 4F :PROBAFELIRAT PRO  
60 :42 41 46 45 4C 49 52 41 54 0D 50 52 4F 42 41 46 :BAFELIRAT PROBAF  
70 :45 4C 49 52 41 54 0D 50 52 4F 42 41 46 45 4C 49 :ELIRAT PROBAFELI  
80 :52 41 54 0D 00 FF 00 00 00 00 00 00 00 00 00 :RAT  
90 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
A0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
B0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
C0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
D0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
E0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :  
F0 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 :
```

A listából kiolvasható a szöveges rekordok szerkezete (22. ábra).

Szöveg	RETURN
--------	--------

22. ábra Szöveg tárolása

Vagyis a szövegben lévő karakterek lemezkódjai után itt is egy 0D (13) RETURN kód áll az adatrekordok elválasztására.

Megjegyzés: a Disk Monitor program lehetőségei közül szemléltetésként egyet bemutatunk. Ha van egy lemezünk, amelyet véglegesíteni szeretnénk, úgy, hogy nem is szándékozunk többé már erre a lemezre írni, akkor két dolgot tehetünk:

- A lemez oldalán levő négyszögletes rést leragasztjuk. Így a lemezegység elektronikája olyan információt kap, amely hardveres úton megakadályozza az írási műveletet. Ha azonban valaki a ragasztást feltépi, máris felülírhatja a programjainkat.
- Emlékezzünk a lemez BAM-jára!

TRACK 18 SECTOR 0

```
00 :12 01 41 00 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F : A  
10 :15 FF FF 1F 15 FF FF 1F 15 FF FF 1F 15 FF FF 1F :  
11 11 11 11
```

A második byte-on egy A betű kódja található, \$41. Ha ezt megváltoztatjuk, és visszafírjuk a lemezre, akkor a DOS ezt a lemezt csak olvasni fogja tudni, írni többé nem! A lemezre való minden írási kísérletnél a:

73,CBM DOS V2.6 1541,18,01

hibaüzenetet kapjuk, jelezvén, hogy a DOS rossz formátumú lemezt talált. Az ilyen lemezt biztonságosan és hibátlanul tudja olvasni, de írni nem tud rá. Jól gondoljuk meg, hogy melyik lemezünket zároljuk így, mert a lemez szerkezetét már csak újrafarmálással tudjuk írásra alkalmassá tenni.

Végezetül néhány, a lemezkezeléssel kapcsolatos trükköt mutatunk be.

Ha egy meghatározott programot keresünk, de nem tudjuk, hogy melyik lemezen van, akkor a tartalomjegyzék egy részének betöltéséhez a következő utasítást használhatjuk.

```
LOAD"$0:prog.nov",8
```

Ha a keresett program nincs a lemezen, akkor csak a lemez neve jelenik meg a képernyőn. Az utasításban a * karaktert is használhatjuk. A

```
LOAD"$0:ABR*",8
```

hatására a lemezen található összes ABR-ral kezdődő programnév kilistázódik.

Ha egy programot betöltés után azonnal indítani is akarunk, akkor ezt egy utasítással is megtehetjük.

```
LOAD"ABRA",8: SHIFT+RUN-STOP
```

A normál LOAD megadása után a RETURN helyet a SHIFT+RUN-STOP billentyűket kell lenyomni a kettőspont után. Ekkor megjelenik a LOAD felirat a képernyőn, és a megnevezett program betöltődik, majd automatikusan indul.

A tartalomjegyzék betöltésénél a megjelöltetni kívánt file-típus is megadható a LOAD utasításban:

```
LOAD"$*=T",8
```

A T helyére a következő paraméterek írhatók:

T	file-típus
P	program
S	soros file
R	relatív file
U	user file

Ez a módszer az összes lemezparancsra igaz. Tehát pl. ha az összes programfile-t le akarjuk törölni a lemeztől úgy, hogy a többi file sértetlen maradjon, a következőket kell megadni:

```
OPEN1,8,15,"S:* =P"  
CLOSE1
```

Hatására az összes programfile törlődik.

6. FEJEZET

Bevezetés a gépi kódú programozásba

Ez a fejezet azokhoz szól, akik már jól elsajátították a BASIC programozási nyelvet, s szeretnének megismerkedni a gépi kódú programozással. Teljességre sajnos a könyv terjedelme miatt nem törekedhetünk. A témát számos, magyar nyelven is hozzáférhető szakkönyv részletesen tárgyalja, pl. a Novotrade kiadásában megjelent *Gépi kódú programozás a C 64-esen* vagy a *Gépi kódú programozás haladóknak*. Az érdeklődők ebből a két könyvből részletesen megismerkedhetnek a gépi kódú utasításokkal, s kitűnő alkalmazási példákat is találhatnak bennük.

Ezt a fejezetet inkább kedvcsinálónak szántuk, és azok figyelmébe ajánljuk, akik, még nem mertek belefogni a gépi kódú programozásba, mert az bonyolultnak, misztikusnak tűnt.

Azok, akik már írtak komolyabb BASIC programokat, bizonyára használták SYS BASIC utasítást. Nos, mint ezt az utasítás leírásánál láthattuk, hatására a C 64-es egy gépi kódú programot hajt végre, amely az utasítás mögé írt címen kezdődik. Írjuk be, s hajtassuk végre a

```
SYS 64738
```

parancsot! A parancs végrehajtásakor a számítógép a bekapcsolás utáni állapotba kerül.

Nézzük meg egy kicsit részletesebben, hogy mi is történt! A tár 64738-as címen levő gépi kódú programot hajtottuk végre, amely különböző feladatok elvégzése után a számítógépünket a bekapcsolási alaphelyzetbe hozza. Ezt a programot nem mi írtuk, ez a számítógépben bekapcsolás után már megtalálható, ui. az operációs rendszer része. Tehát hatalmas gépi kódú program birtokában vagyunk a számítógép bekapcsolásakor anélkül, hogy egy sort is leírtunk volna.

Az operációs rendszer azonban még számos más hasznos rutint is tartalmaz, amelyeket felhasználhatunk a BASIC programjainkban. Csak tudnunk kell, hogy hol kezdődnek, s természetesen azt, hogy mit csinálnak. A jó felhasználáshoz pedig azt is, hogy hogyan!

Ha egy gépi kódú programot nem úgy szeretnénk végrehajtani, ahogy azt a tárban találjuk, hanem pl. más paraméterekkel, akkor ezt is megoldhatjuk a BASIC POKE utasítással. Valószínű, hogy már sokan használták a

```
POKE 53281,7
```

utasításhoz hasonló egyszerű utasítást, amely a képernyő háttérszínét változtatja meg, esetünkben pl. sárgára. A képernyő színének figyelését is egy gépi kódú programrész végzi, amelyet igen sokszor futtat le az operációs rendszer. A színkód paramétereket egy meghatározott helyről (és mindig ugyanarról a helyről) olvassa ki ez a rutin. Ha ezt az értéket átírjuk, akkor a következő kiolvasásnál már a mi általunk

megadott kódot fogja felhasználni. A figyelmes Olvasó bizonyára észrevette, hogy itt egy olyan rutint használtunk fel, amelynek nem tudjuk a kezdőcímét, csak azt a helyet ismertük, amelyhez az rendszeresen fordul. Ebből arra következtethetünk, hogy vannak olyan gépi kódú programok, amelyeket állandóan lefuttat az operációs rendszer, s ezek a rutinok a tár meghatározott rekeszein keresztül tartják a kapcsolatot a „külvilággal”. Az is logikus, hogy ezek a rekeszek mindig ugyanazon a helyen vannak (fix címük van), hiszen így az operációs rendszernek nem kell a cím meghatározásával „bajlódnia”, s a végrehajtásuk is gyorsabb. Szükség is van a „takarékoságra”, hiszen számítógépünk ezeket a programrészeket másodpercenként többször is végrehajtja (minden 0.016 s-ban).

Ha a tár egy meghatározott rekeszének a tartalmára vagyunk kíváncsiak, akkor azt már jól ismert PEEK utasítással olvashatjuk ki. Pl. annak a színnek a kódját, amely a képernyőre írunk, a

PRINT PEEK(646)

utasítással kérdezhetjük le.

Mint azt a példából is láthattuk, ha valaki már használta a SYS, POKE, PEEK BASIC utasításokat, akkor egy kicsit már gépi kódban is programozott. Nincs tehát semmi ok arra, hogy bárki idegenkedjen ettől a témától. Az ismerkedés különösen akkor válik izgalmassá, ha kiderül, hogy mi mindent lehet megoldani egy egyszerű kis gépi kódú programmal, olyan feladatokat is, amelyeket BASIC-ből csak lassan, nehezen vagy egyáltalán nem lehet megoldani.

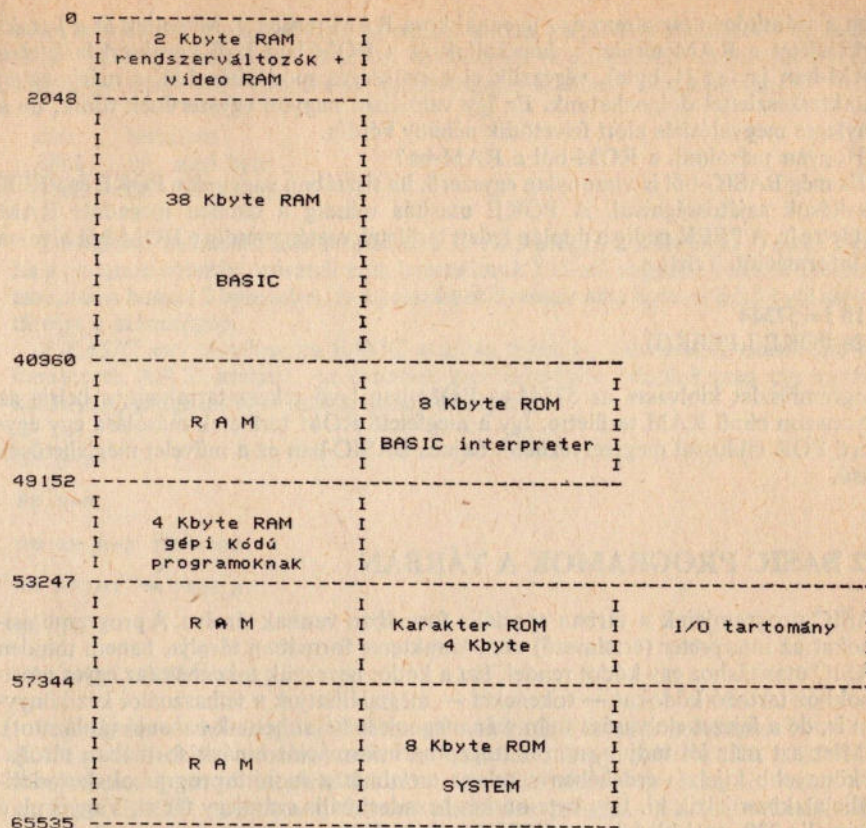
Ne ugorjunk azonban rögtön fejést a gépi kódú programozás rejtelseibe, főleg annak felhasználásába, mert könnyen bajt is okozhatunk magunknak! Az előző példákban ugyan nem emeltük ki eléggé, de talán nyilvánvaló, hogy a kívánt hatás eléréséhez pontosan kell ismernünk a SYS vagy a POKE utasítással kapcsolatos tárcímeket, s ezeket gondosan és hibátlanul kell beírunk az egyes utasításokba.

Ha a képernyő háttérszínét akarjuk megváltoztatni (pl. sárgára), s a POKE 53281,7 helyett a POKE 53280,7 utasítást adjuk ki, akkor nem a háttér, hanem a keret színe fog megváltozni. Ez csak kellemetlen, de nem okoz végzetes bajt. Ne feledjük, ennél nagyobb bajt is okozhatunk programunkban egy rossz POKE vagy SYS utasítással (pl. blokkolhatjuk úgy az operációs rendszert, hogy abból csak a számítógép ki- és bekapcsolásával lehet kizökkenteni, elvesztvén a beírt felhasználói programot). Alaposan nézzünk utána a tárcím jelentésének, mielőtt a tartalmát módosítanánk!

Könnyebben eligazodunk a számítógépünk egyes rekeszei között, ha megnézzük, hogy a gép készítői hogyan „parcellázták” fel a tárat.

6.1 A COMMODORE 64-ES TÁRTÉRKÉPE

A Commodore 64-es típusú számítógép RAM területe 64 kbyte. A bekapcsolás után azonban azt látjuk, hogy csak 38911 byte-ot használhatunk BASIC programjaink számára, vagyis ennyi a szabad tárterület (38 kbyte). Mire fordítódik a fennmaradó 26 kbyte terület? Nos, ebből 20 kbyte-ot az operációs rendszer foglal el, kb. 2 kbyte kell a video RAM munkaterületének, s a maradék 4 kbyte-ot felhasználhatjuk a gépi kódú programjainkhoz vagy adatterületnek. Így valójában 42 kbyte felhasználói RAM területtel rendelkezünk. A 23. ábra a C 64-es alaptárképezését szemlélteti egy kicsit részletesebben.



23. ábra A C 64-es tártérképe

Az ábrából kitűnik, hogy vannak olyan (duplán fedett) RAM területek, amelyeket a számítógép ROM területként tart nyilván, sőt olyan területek is (53247-estől 57344-esig), amelyek háromszorosan fedettek. Ez azt jelenti, hogy ezekben a címtartományokban egy ROM és egy RAM tároló is található. Másféleképpen kifejezve pl. az 57344-es címnek két megfelelője is van: az egyik a ROM, a másik a RAM területen fekszik. A számítógép természetesen csak az egyik tárolótartományt tudja megcímezni. Bekapcsolás után csak az 57344-es rekesz ROM területen levő tartalma létezik, a RAM területi megfelelője elérhetetlen. Ha ezen a tárfelosztáson változtatni szeretnénk, akkor az 1-es cím tartalmát kell módosítanunk.

Miért kellene ezen a jól bevált tárkiosztáson módosítani? Pl. azért, mert valaki más operációs rendszerrel szeretne dolgozni (pl. CP/M-mel), vagy — s ez gyakran előfordul eset —, a karakterkészleten kívánunk változtatni (pl. szeretnénk azt kiegészíteni a magyar ékezetes betűkkel).

Nos, a ROM területen levő karakterkészlet nem módosítható egyszerűen egy programból, hiszen a ROM terület csak olvasható. A probléma mégis megoldható, hi-

szen a számítógép tartalmaz egy ugyanakkora RAM területet. Másoljuk át a karakterkészletet a RAM területre, kapcsoljuk át a ROM-ban lévő karakterkészletet a RAM-ban levőre (1. byte), végezzük el a szükséges módosításokat, s máris az új karakterkészlettel dolgozhatunk. Ez így valójában nagyon egyszerűnek tűnik, de a tényleges megvalósítás előtt felvetődik néhány kérdés.

Hogyan másolunk a ROM-ból a RAM-ba?

Ez még BASIC-ből is viszonylag egyszerű, ha tisztában vagyunk a POKE és PEEK utasítások sajátosságaival. A POKE utasítás mindig a címben megadott RAM területre ír. A PEEK pedig a duplán fedett területek esetén mindig a ROM-ból olvassa az információt. Tehát a

```
10 I=57344
```

```
20 POKE I,PEEK(I)
```

programrészlet kiolvassa az 57344-es ROM-ban levő rekesz tartalmát, s beírja az ugyanazon című RAM területre. Így a megfelelő ROM területek másolása egy egyszerű FOR ciklussal megszervezhető. Sajnos BASIC-ban ez a művelet meglehetősen lassú.

6.2 BASIC PROGRAMOK A TÁRBAN

BASIC programjaink a tárban speciális formában vannak tárolva. A programutasításokat az interpreter (értelmező) nem karakteres formában tárolja, hanem minden BASIC utasításhoz egy kódot rendel. Ezt a kódot nevezzük tokennek (az egyes utasításokhoz tartozó kódokat — tokeneket — megtalálhatjuk a felhasználói kézikönyvben is, de a fejezet elolvasása után már magunk is készíthetünk hasonló táblázatot).

Mint azt már jól tudjuk, a számítógép az információt bináris formában tárolja. A könnyebb kijelzés érdekében a rekesz tartalmát a monitorprogramok hexadecimális alakban jelzik ki. Egy byte-on két hexadecimális számjegy fér el, Vagyis pl. a decimális 230, ami binárisan 11100110-nak felel meg, azaz:

```
11100110
E 6
```

hexadecimális alakban E6!

A BASIC program a 2048-as címen kezdődik (hex. 800). Ezen a címen mindig egy 0 byte található. Az egyes BASIC sorok tárbeli képe a 24. ábrán látható.

I	mutató	I	Sor-	I	0	I	BASIC	I	0	I
I	(LINK)	I	szám	I	byte	I	sor	I	byte	I
I		I		I		I		I		I

24. ábra BASIC program a tárban

A *mutató* (vagy LINK byte): a következő BASIC sorszerkezet elejére mutat, tehát egy tárcím, amelyet a számítógép alsó byte — felső byte alakban tárol. A két byte-os címek úgy vannak a tárban elhelyezve, hogy az alacsonyabb helyiértékű byte (alsó

byte) megelőzi (az alacsonyabb tárcímen) a cím magasabb helyiértékű byte-ját (a felső byte-ot), amely a rá következő rekeszben (a magasabb tárcímen) található. Vagyis pl. a hex. 0809-es cím a következőképpen található meg a tárban:

cím	tartalom
0801	09 alsó byte
0802	08 felső byte

A *sorszám*: hexadecimális formában a BASIC program sorszáma. Hossza 1 byte, ha a program sorszámozásánál nem használtunk 255-nél nagyobb értéket. Különben maximális hossza 2 byte lehet, és a sorszámot ilyenkor alsó byte — felső byte alakban tárolja a számítógép.

A *BASIC sor*: tartalmazza BASIC utasítás tokenjét, valamint az utasításba beírt karakterek ASCII kódjait. Az előbbieket szemléltetésére nézzünk meg egy egyszerű kis BASIC programot, és annak tárbeli elhelyezését.

```
10 A=3
20 B=A
30 IF A=3 THEN 60
40 PRINT "A NEM 3"
50 STOP
60 PRINT "A=3"
```

A program tárbeli megtekintéséhez számos kitűnő monitorprogramot készítettek. A könyv terjedelme azonban nem teszi lehetővé, hogy ezekre külön is kitérjünk. Kezdetben nagyon jól megfelel a közismert HELP+ segédprogram is. Ennek révén nemcsak BASIC programot, hanem assembler programokat is szerkeszthetünk! Hívjuk be, majd indítsuk el a HELP-et! Gépeljük be az előbbi BASIC programot! A] billentyű lenyomásával térjünk át ún. monitorüzemmódba. A képernyőn megjelenik az aktuális tárcím és tartalma. Alapesetben a:

cím	tartalom
0000	2F

látható. A címet úgy tudjuk gyorsan megváltoztatni, ha lenyomjuk a + billentyűt, majd ezután beírjuk a kívánt cím értékét, ami pl. 0800 (a címet és tartalmát mindig hexadecimális alakban kell megadni! Ezt a továbbiakban külön nem jelezzük!).

A képernyőn a következőket kell látnunk:

cím	tartalom
0800	00

Ha lenyomjuk a RETURN billentyűt, akkor a számítógép a következő rekesz tartalmát írja ki. Nézzük végig a tárnak a 0800-astól a 0845-ösig terjedő részét! Ha pontosan gépeltük be a BASIC programot, akkor a következő képet kell kapnunk. A rekeszek tartalmát némi gyakorlás után könnyen visszafejthetjük. Ezt mi is megtettük, s karakteres formában a rekeszek tartalma mellé odaírtuk.


```

0800 00
0801 09 link alsó byte
0802 08 link felső byte
0803 0A 10
0804 00
0805 41 A
0806 B2 =
0807 33 3
0808 00 mindig 0 byte
0809 11 link alsó byte
080A 08 link felső byte
080B 14 20
080C 00
080D 42 B
080E B2 =
080F 41 A
0810 00 0 byte
0811 20 link alsó byte
0812 08 link felső byte
0813 1E 30
0814 00
0815 8B IF
0816 20 SPACE
0817 41 A
0818 B2 =
0819 33 3
081A 20 SPACE
081B A7 THEN
081C 20 SPACE
081D 36 6
081E 30 0
081F 00 0 byte
0820 30 link alsó byte
0821 08 link felső byte
0822 28 40
0823 00
0824 99 PRINT
0825 20 SPACE
0826 22 "
0827 41 A
0828 20 SPACE
0829 4E N
082A 45 E
082B 4D M
082C 20 SPACE
082D 33 3
082E 22 "
082F 00 0 byte
0830 36 link alsó byte
0831 08 link felső byte
0832 32 50
0833 00
0834 90 STOP
0835 00 0 byte
0836 42 link alsó byte
0837 08 link felső byte
0838 3C 60
0839 00
083A 99 PRINT
083B 20 SPACE
083C 22 "
083D 41 A

```

```

083E 3D =
083F 33 3
0840 22 "
0841 00 0 byte
0842 00 link alsó byte
0843 00 link felső byte
0844 00
0845 00
0846 00

```

A lista alapján világos, hogy a LINK byte-ok hogyan mutatnak a következő végrehajtandó utasítás elejére (ha a LINK byte értéke nulla, akkor a program végrehajtása megszakad, l. a 0842-es és a 0843-as címeket).

Az elmondottakból a figyelmes Olvasó már bizonyára rájött, hogy a BASIC utasítások sorszámaira a program végrehajtása közben nincs is szükség. A bemutatott program alapján készíthetünk egy olyan programot, amelyben minden sorszám azonos, és mégis hibátlanul fut! (Természetesen az ilyen programon javítani már igen körülményes lenne). Az egyes címek tartalmát a következőképpen tudjuk módosítani:

a) álljunk rá (a + billentyű és cím beírásával) a módosítandó rekesz címére;
b) nyomjuk le a / (ferde törtvonal) billentyűt, jelezvén, hogy a rekesz tartalmát, is nem a címét kívánjuk beírni;

c) írjuk be a rekesz új tartalmát hexadecimális alakban!

Ne feledjük tehát, hogy BASIC programjaink nem gépi kódban, hanem tokenizált formában vannak tárolva. Az igazi gépi kódú programozással a következő fejezetben smerkedhetünk meg.

6.3 A GÉPI KÓDÚ PROGRAMOZÁS ELŐNYEI

A legtöbb személyi számítógép programozható BASIC nyelven. Sőt, ma már a C 64-eshez is kaphatók különböző fordítóprogramok, amelyekkel programozhatunk FORTRAN, PASCAL vagy C nyelven is. Ezek a programnyelvek a magas szintű programnyelvek csoportjába tartoznak. Azért találtak ki ennyiféle programnyelvet, mert az egyik probléma az egyik nyelven, a másik a másikon oldható meg hatékonyabban. Így fejlődtek ki az ún. problémaorientált programnyelvek. Vannak olyan magas szintű nyelvek, amelyek az adatfeldolgozást támogatják (pl. COBOL), de olyanokkal is találkozhatunk, amelyek a műszaki tudományos feladatok elvégzésére készültek (pl. FORTRAN). Ezek klasszikusként emlegetett, ún. feladatorientált programnyelvek. A C 64-es gépiükön is találhatunk a különböző feladatokhoz speciális nyelveket, BASIC kiegészítéseket, pl. adatfeldolgozáshoz a MASTER 64 vagy a SUPERBASE nevű programokat, nem beszélve a grafikák elkészítését támogató seregnyi BASIC bővítésről!

A gépi kódú programozással is speciális feladatokat tudunk megoldani. Tehát a gépi nyelv is problémaorientált programnyelv! Olyan feladatokat oldunk meg ezen a nyelven, amelyeket nagyon gyorsan kell az adott számítógépen végrehajtani, vagy a rendelkezésre álló tárkapacitás nagyon kicsi. Ha olyan feladatot szeretnénk végeztetni a számítógéppel, amelyet egyetlen magas szintű nyelven sem tudunk vele közölni, akkor is gépi kódú programozáshoz kell folyamodnunk (ilyenek pl. a megszakításokkal kapcsolatos feladatok). A gépi kódban megírt programok tökéletesen képesek igazodni az elvégezendő feladatokhoz.

A sok előnyös tulajdonság ellenére azonban a gépi kódú programozásnak is vannak hátrányai. A gépi kódú programokat nem vihetjük át egyszerűen egy másik gépre, mint pl. egy C nyelvű programot. A gépi kódú programozási nyelv egyik hátránya a processzororientáltság. A programokat csak arra a gépre tudjuk átvinni, amelynek ugyanaz a processzortípusa van, mint amelyen a programot megírtuk. További hátránként vehető, hogy a gépi kódú programokat nehezebb tesztelni, mint pl. egy BASIC programot. Ezért sokan a főprogramot valamilyen magas szintű nyelven írják, a kritikus részeket pedig gépi kódban. Mielőtt rátérnénk a gépi kódú programozásra, tisztáznunk kell egy idáig is kerülgedett tényt! Gépi kódban ma már senki sem ír programot! Helyette inkább az ASSEMBLER nyelvet alkalmazzák! A kettő között a végeredményt tekintve nincs különbség, hiszen az assemblerben írt program minden sora egy gépi kódú utasításnak felel meg. A programot megírni, javítani, tesztelni azonban lényegesen könnyebb assemblerben, mint gépi kódban! Ennek szemléltetésére nézzünk egy egyszerű példát! A HELP-pel menjünk monitor-üzemmódba (J billentyű), majd válasszunk ki egy kezdőcímet, pl. 0840. Nyomjuk le a / (ferde törtvonal) billentyűt (a címek tartalmát fogjuk módosítani), s írjuk be sorban a következő értékeket:

```
0840 AS 159
0841 01 1
0842 0D 111
0843 00 0
0844 04 4
0845 AS 159
0846 00 0
0847 0D 111
0848 00 0
0849 0B 216
084A 4C 16
084B 4A 74
084C 0B 8
```

Ezzel beírtuk az első gépi kódú programunkat, amely a 2112-es címen kezdődik (hex. 0840). Indítsuk is el a SYS 2112 paranccsal! A program a képernyő bal felső sarkába egy fekete A betűt ír. A programot a RUN/STOP RESTORE billentyűk leütésével állíthatjuk meg.

A program beírása jóval nehezebb volt, mint ugyanezt BASIC-ben írtuk volna. Nem beszélve arról, hogy e számhalmazról első ránézésre nem tudjuk eldönteni, hogy melyik az utasítás és melyik az adat. Az egyes utasítások gépi kódjait is nehéz lenne fejben tartani. Nézzük meg, hogy hogyan fest az előbbi program ASSEMBLER kódban:

```
*=0840
LDA #01
STA 08400
LDA #00
STA 08000
C1 JMP C1
```

Első ránézésre is sokkal áttekinthetőbb, világosabb, mint az előző gépi kód (részletesen l. a 6.7 alfejezetben). Ezt a példát azonban mélyebben még ne elemezzük, előbb még néhány dolgot tisztázni kell.

6.4 A 6510-ES MIKROPROCESSZOR REGISZTEREI

Mielőtt megismerkednénk a géi kódú programozással, vizsgáljuk meg közelebbről a mikroprocesszor belső felépítését!

A C 64-es gépek mindegyikében a 6510-es mikroprocesszort találjuk. A processzor ún. regisztereket tartalmaz, amelyekben a műveletek lezajlanak. A regiszterek elektronikusan is kiemelt átmeneti tárolók, amelyek egyik fő jellemzője az, hogy rendkívül gyorsak, hiszen egyetlen utasítás végrehajtása közben többször is változhat a tartalmuk. A 6510-es processzor összesen hat regisztert tartalmaz. Ezek között van öt nyolc- és egy tizenhat bites regiszter.

Akkumulátor: a processzor legfontosabb regisztere. Egy nyolc bites általános munkaregiszter, amelyet a processzor minden aritmetikai, logikai művelet végrehajtásakor igénybe vesz.

Indexregiszterek: tömbök, táblázatok feldolgozása során mutatóként használjuk. Két indexregiszterünk van, X és Y néven.

Veremmutató: a veremtároló mutatója. A veremtár a \$0100—\$01FF (256—511) fix tárcímeken helyezkedik el. Ezt az alprogramok használják átmeneti tárolóként. Jellemzője, hogy az utoljára beírt adatot tudjuk elsőként kiolvasni, az utolsó előtti másodszorra stb., ezért is nevezik veremtárolónak.

Státusz- vagy állapotregiszter: tartalma az utoljára végrehajtott utasítás eredményétől függ. Minden bitje egy ún. jelző (flag) szerepét tölti be. A programban ezeket lekérdezhetjük, s az esetleges elágazások során figyelembe vehetjük. Egyes bitjeit a számítógép automatikusan a művelet elvégzése után állítja be (a táblázatban G-vel jelölve), de vannak olyan bitjei is, amelyeket a programozó is beállíthat (P-vel jelölve).

Bit	Név	Funkció	Ki állítja
0	C	átvitel történt aritmetikai művelet esetén	G—P
1	Z	eredmény nulla	G
2	I	megszakítás engedélyezett-e	G—P
3	D	ha az aritmetika decimális alakban	P
4	B	BRK utasítás lehet	G
5	—	nem használt	
6	V	túlcsordulás az előjeles számoknál	G—P
7	N	az eredmény negatív (nagyobb, mint 127)	G

Utasításszámláló regiszter: a soron következő utasítás címét tartalmazza. Az egyetlen 16 bites regisztere a processzornak. A tár minden egyes rekeszének egy meghatározott címe van, amely tulajdonképpen egy bináris szám. Ha a processzor a címeket is csak 8 biten tárolná, akkor mindössze csak 256 rekeszt tudna megcímezni! Ezért működik a számítógépünk 16 bites címregiszterrel. Így a C 64-es 65536 tárolórekeszt tud megcímezni (64 kbyte).

Ahhoz, hogy a mikroprocesszor dolgozni tudjon, egy programot és az ahhoz esetleg szükséges adatokat el kell helyezni a számítógép tárában. A processzor számára az utasítás és az adat egyaránt egy 0—255 közé eső szám. Azt, hogy melyik az adat és melyik az utasítás, a programozónak kell tudnia. A gépi kódú utasításokat a processzor értelmezi és azonnal végre is hajtja (l. még a 6.6 alfejezetet).

6.5 CÍMZÉSMÓDOK

Nyolc biten összesen 256 féle állapotot tudunk ábrázolni. A gépi kódok számára azonban nincs minden variáns kihasználva. A 6510-es processzor ebből 151-et tekint utasításkódnak. A 151 utasítás között sok csak címezsmódban különbözik egymástól.

Példaként tekintsük azt az utasítást, amely egy megadott tárcím tartalmát tölti egy regiszterbe:

- az LDA utasítás az akkumulátorba;
- az LDX utasítás az X indexregiszterbe;
- az LDY utasítás az Y indexregiszterbe!

Az utasítások végén levő A,X,Y betűk általában az előbbi regiszterek valamelyikére utalnak.

Egy konstans értéket az akkumulátorba a következőképpen tölthetünk:

LDA #12

A konstansok jelölésére a # jelet használjuk a szám előtt, s így az operandust mint számértéket kezeli a processzor. Az utasítás jelentése tehát: 12-t tölts az akkumulátorba! Az itt megadott érték decimális 12! Ha hexadecimális konstans kívánunk beírni, azt a \$ jelnek kell megelőznie.

LDA #\$12

Vagyis: hex. 12-t tölts az akkumulátorba! Ezt a címezési módot, az utasítás jellegénél fogva, közvetlen címezésnek nevezzük. Ha egy tárcímen levő tartalmat szeretnénk mondjuk az akkumulátorba tölteni, akkor azt többféleképpen is megtehetjük. A legegyszerűbb esetben az utasításban adjuk meg a kívánt rekesz címét, pl.:

LDA \$D400

Az utasítás a D400-as címen lévő rekesz tartalmát tölti be. Ezt a címezési módot abszolút címezési módnak nevezzük, hiszen az utasításban abszolút cím van megadva. Ez az utasítás a tárból három byte-ot foglal el. Az első az LDA kódja, a következő két byte-on pedig a cím található, alsó byte — felső byte formában. Van ennek a címezési módnak egy speciális alakja, a nulláslap (zero page) címezés, amely, a 6510-es mikroprocesszor egyik sajátossága. Akkor használjuk, ha a cím kisebb mint 255. Az elnevezés onnan ered, hogy a tár 256, egyenként 256 byte-os területre (ún. lapokra) van felosztva.

Tehát az:

LDA \$A3

utasítás tárolásához csak két byte-ra van szükség, mivel a nulláslapon levő cím (amely legfeljebb 255) felső byte-ja mindig nulla. Azon kívül, hogy helyet is takarítunk meg, ez az utasítás gyorsabban hajtódik végre, mint három byte-os változata.

Ha a tár egy meghatározott címétől kezdődően folyamatosan szeretnénk pl. adatokat tölteni, akkor az indexelt címezsmódot használhatjuk. Az indexelést az X vagy az Y indexregiszter segítségével végezhetjük el. Pl.:

LDA \$C400,X

utasításban a processzor nem a C400-as rekesz tartalmát fogja betölteni, hanem előtte ehhez hozzáadja az X regiszter tartalmát. Vagyis, ha az X regiszter tartalma pl. \$15, akkor a $\$C400 + \$15 = \$C415$ című rekesz tartalmát fogja betölteni. Ezt az indexelési módot kitűnően használhatjuk pl. tömbök feldolgozása során. Természetesen a példánkban X helyett használhattuk volna az Y indexregisztert is. E tekintetben a két regiszter teljesen egyenrangú.

Az indexelt címezésnek van egy másik formája, amely ugyan egy kicsit bonyolultabb, de igen hatékony: az indirekt indexelt címezés. Tekintsük a következő utasítást:

LDA (\$A3),Y

Mint látjuk, a címezés első felében a nulláslap egy rekeszére hivatkoztunk. Itt található a 16 bites cím alsó byte-ja. A felső byte a következő címen van (\$A4-en). A processzor erről a területről veszi a címet, s ehhez még hozzáadja az Y indexregiszter értékét. Tehát, ha pl. a \$A3 tartalma 00, a \$A4 tartalma C4, így a címszámítás alapja a \$C400 és ha Y-ba \$12 van, akkor a $\$C400 + \$12 = \$C412$ -es rekesz tartalmát fogjuk betölteni. A címezsmód rendkívül hatékony, mivel egy 2 byte-os utasítással egy 16 bites címet tudunk kezelni. Formailag arról lehet könnyen felismerni, hogy az egyik operandusa zárójelben van. Az utasítást csak az Y indexregiszterrel és nulláslap címmel használhatjuk!

Talán az egyik legnehezebben követhető címezsmód, s egyben a legutolsó is, az indexelt indirekt címezés. Itt csak az X regiszterrel dolgozhatunk. Nézzünk erre is egy példát:

LDA (\$A0,X)

A cím kiszámításánál megint a nulláslap egyik rekeszét használjuk. A különbség az előző esethez képest az, hogy a \$A0-hoz adjuk most X tartalmát, s az ezen a címen levő érték lesz a leendő cím alsó byte-ja. A felső byte értékét is hasonlóan kapjuk meg. $\$A0 + 1$ -hez hozzáadjuk X tartalmát, s ezen a címen levő értéket vehetjük a cím felső byte-jának. Vegyünk egy konkrét példát! Ha az X tartalma \$03, akkor:

$\$A0 + \$03 = \$A3$

\$A3-ban \$00 a cím alsó byte-ja

\$A4-ben \$C4 a cím felső byte-ja

Vagyis a C400-as rekesz tartalmát fogjuk betölteni. Ezzel a címezési módok tárgyalását befejeztük.

A következő fejezetben néhány példán keresztül bemutatjuk a gépi kódú programozás technikáját. Amint azt már említettük, az assembler programok elkészíté-

séhez a HELP+ segédprogramot használjuk, abból a megfontolásból, hogy ez a program egyszerűsége és könnyen kezelhetősége alapján az első lépések megtételének legalkalmasabb eszköze.

Azt nem állítjuk, hogy a fejezet áttanulmányozása után az Olvasó a gépi kódú programozás avatott mesterévé válik. Szándékunk mindössze annyi, hogy mindazoknak, akik eddig a gépi kódot misztikusnak és érthetetlen szimbólumok szövevényeként, kellő tisztelettel és lehetőleg távolról figyelték, bebizonyítsuk, hogy a félelem indokolatlan, és eloszlatása csak megfelelő szemlélet- és közelítésmód kérdése. Éppen ezért az utasítások betűrendes leírását csak a mellékletben közöljük és hangsúlyt a nyelv szisztematikus ismertetése helyett az egyszerű, szemléletes példaprogramok bemutatására helyezzük.

Úgy gondoljuk, hogy kellő bátorság és kedv, ill. megfelelő szemléletmód birtokában — és szívből reméljük, hogy a fejezet mindezek megszerzésére alkalmas — már csak kitartás kérdése, hogy egy jó és részletes szakkönyvet bátran a kezünkbe véve, bármilyen nehéznek tűnő feladat gépi kódú programját elkészítsük.

6.6 EGYSZERŰ MINTAPÉLDÁK

Először is tekintsük át részletesebben a 6.3. alfejezet végén ismertetett assembler programot, amely egy fekete A betűt írt ki a képernyő bal felső sarkába. Írjuk meg ezt a programot még egyszer a HELP program segítségével, és fordítsuk le gépi kódra! Fordítás előtt az assembler programot pontosan úgy kell begépelni, mintha BASIC program lenne. Tehát:

```
10 *=$0840
20 LDA #$01
30 STA $0400
40 LDA #$00
50 STA $D800
60 C1 JMP C1
```

A sorszámokra azért van szükség, hogy a BASIC interpreter az egyes utasításokat ne próbálja meg parancsként értelmezni. Elemezzük a sorokat részletesen!

10 *=\$0840

A * parancs segítségével mondhatjuk meg a HELP assembler fordítójának, hogy a következő utasítást melyik tárcímre tegye. Az = jel után mindig egy tárcím következik, a legtöbb esetben hexadecimális alakban (\$-al jelezve), de megadhatjuk a címet decimális alakban is (\$ jel nélkül). Vagyis a:

10 *=\$0840

utasítás a program kezdetét a hex. 840-es címre helyezi (2112 decimális), így a SYS 2112 paranccsal tudjuk indítani. A

10 *=840

utasítás a hex. 348-as címre helyezi a programot, s így azt a SYS 840 paranccsal indíthatjuk.

A * utasítást a program közben többször is alkalmazhatjuk (pl., ha a változók számára fenntartott tárterületet a programon kívül, a tár egy más területére kívánjuk elhelyezni). A következő utasítás:

20 LDA #\$01

az akkumulátorba tölt 1-et — mivel egy A betűt szeretnénk a képernyőre kiírni, s az A betű képernyőkódja (nem azonos az ASCII kóddal): 1. (Természetesen a képernyőkódok is ábécé rendben növekednek). A hexadecimális jelölés, a \$ jel feleslegesnek tűnhet, mivel 01 decimális és hexadecimális alakban is ua. jelenti. Mi azért tettük ki a \$ jelet, mert már a programozás kezdetén nem árt megszokni, hogy gépi kódban az értékeket és a címeket hexadecimális alakban kell megadni, hiszen mint azt már korábban említettük, azokat minden monitorprogram hexadecimálisként jelzi vissza.

A következő utasítás:

30 STA \$0400

az akkumulátor tartalmát kiírja a \$0400-as címre (1024 dec.). Itt kezdődik a képernyőtár, s az itt tárolt információt (1000 byte-ot) a számítógép a képernyő bal felső sarkától kezdve folyamatosan jeleníti meg. Mivel az akkumulátorba az előző utasítással 1-et vittünk, így a 30-as utasítás ezt az értéket írja majd ki. Azonban ahhoz, hogy az 1-es kódnak megfelelő értéket lássuk is a képernyőn, a színtárba a megfelelő helyre egy színkódot is be kell írunk. A kiírt karakter ui. háttér színével fog megjelenni, és pl. kék alapon a kék A betűt senki sem fogja látni (csak akkor, ha esetleg a kurzort arra a helyre viszi). Újabb típusú C 64-es gépeknél erre az utasításra csak akkor van szükség, ha a kiírandó karaktert más színnel kívánjuk kiírni, egyébként a keret színében jelenik meg. A következő utasítással egyszerű eldönteni, hogy a számítógépünk régebbi vagy újabb sorozatból való. A számítógép bekapcsolása után írjuk be közvetlenül ennek a kis gépi kódú programnak a BASIC megfelelőjét!

POKE 1024,1

utasításra a bal felső sarokban egy A betű jelenik meg, akkor biztosak lehetünk benne, hogy a számítógép az újabb sorozatból származik (pl. nagyobb integráltságú VIC IC-t tartalmaz stb.). Mivel mi a bal felső sarokban egy fekete A betűt szeretnénk látni (függetlenül attól, hogy gépünk régebbi vagy újabb), mindenképpen szükségünk van a következő utasításokra:

40 LDA #\$00

Betöltjük az akkumulátorba a fekete szín kódját (0), majd ezt az:

50 STA \$D800

utasítással kiírjuk a színtárba. A színtárról most csak annyit kell tudnunk, hogy

mérete megegyezik a képernyőtár méretével. \$D800-as címen kezdődik, s minden egyes byte-ja a képernyőtár kezdetétől számított ugyanannyiadik byte színét határozza meg.

Az utasítások hatására a bal felső sarokban megjelenik a fekete A betű. Azért, hogy ezt tartósan, folyamatosan lássuk, a programot itt megállítottuk egy ún. végtelen ciklussal:

```
60 C1 JMP C1
```

Ez az utasítás egy feltétel nélküli ugrást hajt végre a C1 szimbólummal jelölt címre. Mivel ez a JMP utasítás eleje, így a programunk ezt fogja ismételni, míg a RUN/STOP RESTORE billentyűket le nem nyomjuk. Az utasítás egyébként megfelel a:

```
10 GOTO 10
```

BASIC utasításnak. Ha ezt az utóbbi utasítást kihagyjuk, akkor programunk „nem szabályosan” fejeződik be. Ezt arról fogjuk észrevenni, hogy a képernyő bal felső sarkában ugyan felvillan az A betű, de a következő pillanatban az egész képernyő törlődik. Helyettesíthetjük a 60-as sort egy RTS utasítással, s így szabályosan térünk vissza a BASIC parancsüzemmodjában. A továbbiakban programjainkat az RTS utasítással fogjuk befejezni.

Miután programunk működését részletesen is áttekintettük, fordítsuk le azt gépi kódra! Ehhez előbb a programot lemezre kell írni, valamilyen néven (pl. G/1). Ha a HELP program már a tárban van, akkor üssük le a [billentyűt! Ezzel hívjuk ui. a HELP assembler fordítóját. Eredeti HELP esetén a következő kérdés jelenik meg a képernyőn:

PROGRAMNAME:

vagyis írjuk be a lefordítandó program nevét! Esetünkben:

G/1

Majd a következő kérdés jelenik meg:

HEXA KORR-POKE:

Ide azt a korrekciós értéket kell beírunk, amennyivel szeretnénk eltérni a *=(cím) utasításban megadott értéktől. Ezt a címmódosító utasítást elég ritkán alkalmazzuk, hiszen a program tárbeli kezdőcíméről a * utasítással már egyszer döntöttünk. Akkor lehet erre esetleg szükség, ha olyan programot kívánunk írni, amelyik a HELP helyén fog futni (\$8000-től), pl. azért, mert egy külső EPROM-ba szeretnénk azt majd beégetni. Ha a címet nem módosítanánk, úgy a lefordított program felülírná a HELP-et, s beláthatatlan zavarokat okozna. (A tapasztalat azt mutatja, hogy ha valaki ilyen programokat ír, akkor már nem a HELP-el dolgozik.) Ha a fordítás során címmódosítást nem kívánunk alkalmazni, vagyis megfelel a * utasításban megadott cím, akkor erre a kérdésre a RETURN billentyűvel válaszoljunk.

A fordító utolsó kérdése:

AUSDRUCK-CODE:

Erre három, a 0 és 1-es számok kombinációiból álló sorozattal felelhetünk. A számok jelentése a megfelelő helyen:

- érték: 1 — a hibás sorokat is kiírja,
0 — a hibás sorokat nem írja ki;
- érték: 1 — a teljes forrásnyelvi programot kiírja,
0 — csak a hibás sorokat listázza ki;
- érték: 1 — szimbólumtáblázatot is készít,
0 — nem kérünk szimbólumtáblázatot.

Ha a fordítás során a nyomtató be van kapcsolva, akkor a lista ott is megjelenik, egyébként csak a képernyőn látható. A legrészletesebb listát akkor kapjuk, ha az utolsó kérdésre 111-gyel válaszolunk.

Program neve	G/1			
Tárcím	Gépi kód	BASIC sorszám	Cím	Assembler kódok
0840		100		*=\$0840
0840	A9 01	110		LDA ##01
0842	8D 00 04	120		STA \$0400
0845	A9 00	130		LDA ##00
0847	8D 00 D8	140		STA \$D800
084A	4C 4A 08	150	C1	JMP C1
084D		65535		.END

A program 7 sorból áll
ZEILEN: 7 SYMBOLE: 1
A hibák száma
FEHLER: 0/

Szimbólumtábla
C1 = 084A

A következő ábra egy hibás sort tartalmazó program fordításakor készült.

```
100 *=$0840
110 LDA ##01
120 STA $0400
130 LDH ##00
140 STA $D800
150 C1 JMP C1
READY.
```

G/2			
0840		100	*=\$0840
0840	A9 01	110	LDA ##01
0842	8D 00 04	120	STA \$0400
	130 SYNTAX ERROR		
0845		130 LDH	##00
0845	8D 00 D8	140	STA \$D800
0848	4C 48 08	150 C1	JMP C1
084B		65535	.END

ZEILEN: 7 SYMBOLE: 2 FEHLER: 1

C1 = 0848 LDH = 0845

SEITE: 1

A 130-as sorban SYNTAX ERROR hibaüzenetet kaptunk. Mivel az utasítást a fordító nem ismerte fel, így nem tudta elkészíteni a megfelelő gépi kódú sort sem.

A lefordított, hibátlan programot elindíthatjuk a SYS paranccsal.

Hosszabb programok esetén, vagy ha BASIC-ből szeretnénk gépi kódú programot hívni, felmerül az igény arra, hogy a már lefordított programunkat lemezen tároljuk és legközelebb (a fordítási fázis kikerülésével) közvetlenül a helyére tudjuk tölteni, ahol az már futtatható. A monitorprogramok tartalmaznak SAVE parancsot, amelynek révén a kész gépi kódú programot vagy (ami ezzel azonos) egy tetszőleges tárterületet lemezeire írhatunk. HELP programunk azonban ezt közvetlenül nem teszi lehetővé, mivel nem tartalmaz ilyen jellegű parancsot. Ezért kis cselhez kell folyamodnunk. Azt már tudjuk, hogy a SAVE parancs (HELP-ben a <- a megfelelője) a BASIC programjainkat lemezeire írja. Nos, ez a parancs a következőképpen működik: a nulláslap \$2B, \$2C (43,44)-es címen található a BASIC program kezdetének a címe, alsó byte — felső byte formában. Hasonló alakban a \$2D, \$2E (45,46)-os címeken a BASIC program vége. A SAVE utasítással ezen címek közti terület tartalmát írjuk a lemezeire. Ha azt szeretnénk, hogy ne a BASIC programunkat, hanem a lefordított gépi kódú programot tároljuk, úgy ezeket a tárcímeket módosítanunk kell. A módosítást elvégezzük a HELP programmal. Tegyük fel, hogy a programunk a \$0840-estől a \$084D-ig található a tárban gépi kódban. Lépjünk be monitorüzemmódba a J billentyű lenyomásával, majd módosítsuk a címet a + billentyűvel 002B-re. Írjuk át a cím tartalmát (/ billentyű) \$40-re (a gépi kódú program kezdőcímének alsó byte-ja)! Nyomjuk le a RETURN billentyűt és hasonló módon írjuk át az előbbi címet! Módosítás után a következőket kell látnunk a képernyőn:

```
002B 40
002C 08      pr. Kezdeté 0840
002D 40
002E 08      pr. vége    084D
```

Lépjünk ki a monitorüzemmódból az = billentyű lenyomásával, majd adjuk ki a már megszokott SAVE parancsot! Így a tárban levő gépi kódú programunkat lemezeire vittük. Győződjünk meg a művelet sikerességéről! Kapcsoljuk ki és be a gépet, hívjuk be a programot a LOAD"pr.név",8,1 paranccsal, s futtassuk le! Ha mindent helyesen végeztünk, akkor a program biztosan jól fog működni.

A HELP segédprogramnak van még egy további hasznos szolgáltatása, amit eddig nem említettünk. A tárbeli gépi kódú programot visszafordíthatjuk assemblerre a HELP programmal, azaz disassemblálhatjuk. Monitorüzemmódban (J) a cím kiválasztása után (+) a HELP a - (kötőjel) billentyű lenyomására dissasemblál egy sort. Ha a szóköz billentyűt nyomjuk le, úgy a kiválasztott címtől folyamatosan végzi a dissasemblálást. Ezt bármikor felfüggeszthetjük a RUN/STOP billentyű egyszeri lenyomásával, s a szóköz billentyű újbóli leütésével folytathatjuk. Kilépés az = billentyűvel lehetséges.

6.7 PÉLDÁK UTASÍTÁSTÍPUSOKRA

A következőkben néhány olyan egyszerű BASIC programrészlet gépi kódú megfelelőjét fogjuk áttekinteni, amelyekből nagyobb programokat is felépíthetünk. Ezeket az elemeket jellegüknél fogva a program építőköveinek is nevezhetnénk. A tárgyalás sorrendjében a következő elemi feladatokat oldjuk meg gépi kódban:

- értékadó utasítás;
- aritmetikai alapl műveletek;
- összehasonlítások;
- programelágazások;
- ciklusok;
- szubrutinok;
- input/output műveletek.

6.7.1 Értékadó utasítások

Mint azt már korábban említettük a processzor számára nem léteznek változók, csak tárrekeszek, ill. azok címei. A

10 X1=18

BASIC utasítás gépi kódú megfelelője, HELP-pel megírva:

```
10 *=$840 ; A PROGRAM KEZDETE
20 X1      = $C400 ;X1 VALTOZO CIME
30 LDA #18 ; DECIMALIS 18 AZ AKKU-BA
40 STA X1 ; AKKU TARTALMA KIIRVA X1 CIMRE
READY.
```

A program fordítási listája:

```
0/3
SEITE:1

0840      10      *=$840 ; A PROGRAM KEZDETE
C400      20 X1    = $C400 ;X1 VALTOZO CIME
0840 A9 12    30    LDA #18 ; DECIMALIS 18 AZ AKKU-BA
0842 8D 00 C4 40    STA X1 ; AKKU TARTALMA KIIRVA X1 CIMRE
0845      65535    .END

ZEILEN:5  SYMBOL:1  FEHLER:0

X1      =C400
```

Ha az előbbi BASIC programot lefuttatjuk, akkor a helyes végrehajtásról úgy győződhetünk meg, hogy kiíratjuk az X1 változó tartalmát. Ha az 18, akkor a program az X1=18 utasítást végrehajtotta. Hasonlóan a gépi kódú program lefutásáról meggyőződhetünk, ha kiíratjuk a \$C400-as rekesz (50176) tartalmát a

PRINT PEEK(50176)

utasítással. De a HELP programmal is megnézhetjük a \$C400-as rekeszt, s ott \$12-t fogunk találni.

6.7.2 Aritmetikai műveletek

További programjainknak célszerű olyan tárterületet kijelölnünk, amelyet a rendszer nem használ, így különösebb bajt nem okoz az ottlétük. A 828-asról a 1019-esig található egy szabad terület, a kazettapuffer. Ha nem használunk kazettás egységet, akkor ez a terület ideális az apró gépi kódú rutinok számára.

a) Összeadás

BASIC programunk legyen a következő:

```
10 X1=18
20 X2=X1+22
```

Assembler megfelelője (HELP-pel beírva):

```
10 *=$33C ; A PROGRAM KEZDETE
20 X1 = $C400 ;X1 VALTOZO CIME
25 X2 = $C401 ;X2 VALTOZO CIME
30 LDA #18 ; DECIMALIS 18 AZ AKKU-BA
40 STA X1 ; AKKU TARTALMA KIIRVA X1 CIMRE
50 LDA #22 ; AKKU-BAN 22
60 CLC ; CARRY FLAG TORLESE
70 ADC X1 ; AKKU=AKKU + X1
80 STA X2 ; AKKU TARTALMA KIIRVA X2 CIMRE
90 RTS
READY.
```

A lefordított program listája:

6/4

SEITE:1

```
033C      10      *=$33C ; A PROGRAM KEZDETE
C400      20 X1    = $C400 ;X1 VALTOZO CIME
C401      25 X2    = $C401 ;X2 VALTOZO CIME
033C A9 12      30    LDA #18 ; DECIMALIS 18 AZ AKKU-BA
033E 8D 00 C4    40    STA X1 ; AKKU TARTALMA KIIRVA X1 CIMRE
0341 A9 16      50    LDA #22 ; AKKU-BAN 22
0343 18         60    CLC ; CARRY FLAG TORLESE
0344 6D 00 C4    70    ADC X1 ; AKKU=AKKU + X1
0347 8D 01 C4    80    STA X2 ; AKKU TARTALMA KIIRVA X2 CIMRE
034A 60         90    RTS
034B         65535    .END
```

ZEILEN:11 SYMBOLE:2 FEHLER:0

X1 =C400 X2 =C401

(A továbbiakban csak a fordítási listákat fogjuk közölni, mivel azok áttekinthetőbbek.) Az X2 változót a \$C401-es címre helyeztük, így a program futtatása után (SYS 828) az eredményt megnézhetjük pl. a:

PRINT PEEK(50177)

utasítással.

A 60-as utasítás törli az átvitelt jelző bitet (carry bit). Az eredmény 1 byte-on elfér, mivel kisebb mint 255. Ha két olyan számot szeretnénk összeadni, amelynek eredménye nem fér el 8 biten, azt az átvitelbit jelzi. Nézzünk erre is egy példát! BASIC programunk legyen a következő:

```
10 X1=78
20 X2=X1+1000
```

A program assemblerben:

6/5-A

SEITE:1

```
C400      10      *=$C400 ; A VALTOZOK KEZDETE
C400      20 X1    = $C400
C402      30 X2    = $C402
033C      40      *=$33C ; A PROGRAM KEZDETE
033C A9 4E      42    LDA #4E ; 78-AS KONSTANS
033E 8D 00 C4    44    STA X1
0341 AD 00 C4    50    LDA X1
0344 18         60    CLC
0345 69 E8      70    ADC #E8
0347 8D 02 C4    80    STA X2
034A AD 01 C4    90    LDA X1+1
034D 69 03      100   ADC #03
034F 8D 03 C4    110   STA X2+1
0352 60         120   RTS
0353         65535    .END
```

ZEILEN:15 SYMBOLE:2 FEHLER:0

X1 =C400 X2 =C402

A programból látjuk, hogy az X1 és az X2 változóknak 2—2 byte-os területet kellett lefoglalni. Az összeadást az alsó byte-ok összeadásával kezdtük.

$$\$4E + \$E8 = \$136$$

Tehát átvitel keletkezett (C=1). A következő ADC utasítás az állapotregiszter C-bitjét (amelynek tartalma 1) is hozzáadja az akkumulátor tartalmához. Mivel az ADC utasítás ezt mindig elvégzi (ha C=0, akkor is), a 60-as sorban levő CLC utasítás feltétlenül szükséges, hiszen ha a carry bitet nem töröljük, akkor az esetleges előző műveletek során megmaradt értéket fogja felhasználni, s ez természetesen hamis eredményre vezethet (ha ui. C előtte 1 volt).

A HELP assemblerben van olyan utasítás, amellyel a változók helyfoglalását és kezdeti értékeit egyszerűbben is megadhatjuk. Ez a .BYTE utasítás. Az operandusmezőben annyi számérték szerepel egymástól vesszővel elválasztva, ahány byte-ot a megadott címtől a címmezőben levő szimbólum számára szeretnénk lefoglalni. A következő programrészlet a .BYTE utasítás felhasználásával készült.


```

C400      10      *$C400 ; A VALTOZOK KEZDETE
C400 4E 00      20 X1      .BYTE 78,0 ;X1 VALTOZO HELYE
C402 00 00      30 X2      .BYTE 0,0 ;X2 VALTOZO HELYE
033C      40      *$33C ; A PROGRAM KEZDETE
033C AD 00 C4      50      LDA X1
033F 18          60      CLC
0340 69 E8          70      ADC #$E8
0342 8D 02 C4      80      STA X2
0345 AD 01 C4      90      LDA X1+1
0348 69 03          100     ADC #$03
034A 8D 03 C4      110     STA X2+1
034D 60          120     RTS
034E          65535      .END

```

ZEILEN:13 SYMBOLE:2 FEHLER:0

X1 =C400 X2 =C402

A program másik érdekessége, hogy a *címmódosító utasítást kétszer is alkalmaztuk.

b) Kivonás

Amikor a számítógépnek egy számból egy másikat kell kivonnia, akkor képezi az utóbbi negatív megfelelőjét (komplementjét), s ezt hozzáadja az előbbihez. Ez azért van így, mert azokból a logikai elemekből (AND, OR, XOR, NOT), amelyek elektronikus megvalósításait számítógépünk processzora tartalmazza, az összeadás és a tagadás (negálás) előállítható, de a kivonás nem!

Negatív számokat úgy ábrázolunk, hogy a legmagasabb helyiértékű bitet (7. bit) 1-re állítjuk. Így előjelesen 1 byte-on $-128 \dots +127$ között ábrázolhatunk számokat. A számok komplementjének képzése (-1 -gyel való beszorzása) a következő algoritmus alapján valósítható meg:

- minden bitet invertálunk (ellenkezőjére állítjuk);
 - majd 1-et hozzáadunk.
- Pl. képezzük 1-ből a -1 -et!

```

00000001 (+1)
11111110 invertálva
+00000001
11111111 (-1)

```

Nézzük meg, hogy kivonáskor az átvitelt jelző bit (carry bit) hogyan változik! Végezzük el az $1 - 1 = 0$ kivonást! Ezt úgy tudjuk csak elvégezni, ha $(+1)$ -hez (-1) -et adunk. Vagyis:

```

00000001 (+1)
+11111111 (-1)
100000000 (0)

```

Az eredmény 0, de sajnos átvitel keletkezett ($C=1$). Ebből azt a következtetést vonhatjuk le, hogy kivonás esetén, ha a $C=1$, akkor nincs átvitel, ha $C=0$, akkor igen. Tehát éppen fordítva mint az összeadásnál. Ezért kivonás előtt a carry bitet mindig 1-re kell állítani a SEC utasítással. Nézzünk erre is egy példát!

A program BASIC nyelven:

```

10 X1=18
20 X2=X1-8

```

A program assembler megfelelője:

```

C400      10      *$C400 ; VALTOZTERULET KEZDETE
C400 12          20 X1      .BYTE 18 ; X1=18
C401 00          30 X2      .BYTE 0 ; X2=0
033C      40      *$33C ; PROGRAM KEZDETE
033C AD 00 C4      50      LDA X1
033F 38          60      SEC ; CARRY-BIT 1-RE ALLITASA
0340 E9 08          70      SBC #$08 ; AKKU=AKKU - 8
0342 8D 01 C4      80      STA X2 ; EREDMENY TAROLASA X2-BEN
0345 60          90      RTS
0346          65535      .END

```

ZEILEN:10 SYMBOLE:2 FEHLER:0

X1 =C400 X2 =C401

Mivel a két byte-os számok kivonásának programja hasonló az előzőekhez, a program különösebb magyarázatot nem igényel.

A BASIC program:

```

10 X1=1078
20 X2=X1-78

```

A program assembler megfelelője

```

C400      10      *$C400 ; VALTOZTERULET KEZDETE
C400 36 04          20 X1      .BYTE $36,04 ; X1=1078
C402 00 00          30 X2      .BYTE 0,0 ; X2=0
033C      40      *$33C ; PROGRAM KEZDETE
033C AD 00 C4      50      LDA X1
033F 38          60      SEC ; CARRY-BIT 1-RE ALLITASA
0340 E9 4E          70      SBC #$4E ; AKKU=AKKU - 78
0342 8D 02 C4      80      STA X2 ; ALSO-BYTE TAROLASA X2-BEN
0345 AD 01 C4      90      LDA X1+1 ; FELSO BYTE TOLTESE
0348 E9 00          100     SBC #0 ; MIVEL A 78 ESETEBEN
                                A FELSO BYTE NULLA
034A 8D 03 C4      110     STA X2+1 ; FELSO BYTE TAROLASA
034D 60          120     RTS
034E          65535      .END

```

ZEILEN:13 SYMBOLE:2 FEHLER:0

X1 =C400 X2 =C402

c) Szorzás, osztás

Mint azt a fejezet elején már említettük, ezeket a műveleteket ismételt összeadásra, ill. kivonásra vezethetjük vissza. Az előző programok felhasználásával ezeket az Olvasó saját maga is könnyűszerrel elkészítheti.

Van azonban a szorzásnak és az osztásnak egy speciális esete, amelyet nem célszerű összeadásra, ill. kivonásra visszavezetni. Ez a kettővel való szorzás, ill. osztás. Ha tízes számrendszerben egy jegyet egy helyiértékkel balra tolunk, az megfelel a szám 10-zel való szorzásának. Hasonlóan, ha a kettes számrendszerben egy bitet egy helyiértékkel balra tolunk, akkor az a kettővel való szorzásnak felel meg. Ezt az ASL utasítással tehetjük meg, amely az akkumulátor tartalmát egy helyiértékkel balra tolja. A „lecsorgó” bitek a carry bitbe kerülnek. Az akkumulátor tartalmának jobbra tolását (2-vel való osztást) hasonló módon az LSR utasítás végzi el.

Bonyolultabb számításokhoz (vagy lebegőpontos számokkal való műveletek esetén) használjuk a BASIC interpreter aritmetikai rutinjait. Erről bővebben a NO-VOTRADE kiadásában megjelent „Gépi kódú programozás haladóknak” c. könyvben olvashatunk.

6.7.3 Összehasonlítások

Programjainkban gyakran van szükség arra, hogy két vagy több változó tartalmát összehasonlítsuk. Vajon hogyan lehetséges ez gépi kódban?

Mielőtt a számítógép pl. az $A=B$ összehasonlítást elvégezné, átalakítja az egyenlőséget $A-B=0$ formára. Így az egyenlőségjel bal oldalán álló kifejezésről azt kell eldöntenie, hogy az nulla-e. Erre azért van szükség, mert a nulla az egyetlen szám, amiről a processzor el tudja dönteni, hogy benne van-e az akkumulátorában vagy sem. Az összehasonlítást úgy végzi el, hogy az akkumulátor bitjei között egyenként egy OR (vagy) kapcsolatot létesít. Ha az eredmény nulla, akkor az akkumulátor minden bitje nulla. Ha az összehasonlításban az $=$ jel helyett $>$ vagy $<$ jel van, akkor a processzor az operandusok kivonása és a bitenkénti OR művelet után az előjelbitből állapítja meg a reláció eredményét.

Írjuk át gépi kódba a következő kis BASIC programot:

```
10 X1=18
20 X2=28
30 IF X1>X2 THEN X3=X1
40 IF X2>X1 THEN X3=X2
```

A program két érték közül a nagyobbát teszi X3 változóba, s egyenlőség esetén X3 tartalma nem változik. (A 10-es és a 20-as sorban levő utasításokat természetesen csak a program kipróbálhatósága kedvéért írtuk be.)

A program assembler megfelelője:

G/B

```
C400      10      *=$C400
C400 1C    20 X1    .BYTE 28
C401 1C    30 X2    .BYTE 28
C402 00    40 X3    .BYTE 0
033C      50      *=$33C
033C AD 00 C4    60    LDA X1
033F      65      ;AKKU - X2 RE VISSZAVEZETETT
                        OSZSEHASONLITAS
033F CD 01 C4    70    CMP X2
0342      75      ;UGRAS HA AZ EREDMENY POZITIV
                        X1>X2 ESETEN
0342 10 06    80    BPL C1
0344      85      ; HA AZ EREDMENY NEGATIV X2>X1 ESETEN
0344 AD 01 C4    90    LDA X2
0347 8D 02 C4   100   STA X3
034A 60      110 C1   RTS
034B      65535     .END
```

ZEILEN:15 SYMBOLE:4 FEHLER:0

C1 =034A X1 =C400 X2 =C401 X3 =C402

Az összehasonlító utasításokra még láthatunk néhány példát a ciklusszervezésnél is.

6.7.4 Programelágazások

Ha az utasításokat valamilyen okból nem a beírás sorrendjében kívánjuk végrehajtani, akkor a program végrehajtását arra a részre kell terelni, amelyik logikailag következik. Ilyen „terelő” utasítások az ugró utasítások. Az ugró utasításoknak, mint azt a BASIC nyelvből már jól ismerjük, két fajtája van:

- feltétel nélküli
 - BASIC megfelelője GOTO (sorszám)
 - assemblerben JSR (cím)
- feltételes
 - BASIC megfelelője IF (log.kif.) THEN (sorszám)
 - assemblerben BEQ, BNE, BCS, BCC, BMI, BPL, BVS, BVC

A feltétel nélküli elágazás annyira egyszerű (a továbbiakban is gyakran előfordul), hogy erre külön példát nem mutatunk be. A feltételes elágazásra már az előzőekben már láttunk egy példát.

6.7.5 Ciklusok

Az egyik leggyakrabban használt program-építőelem. Írjuk be a következő kis BASIC programot:

```
10 KEP=1024
20 SZIN=55296
30 Y=0
40 POKE KEP+Y,1
50 POKE SZIN+Y,0
60 Y=Y+1
70 IF Y=160 THEN STOP
80 GOTO 40
```

A program 160 fekete A betűt ír a képernyőre (a képernyő első 4 sorába). A ciklust megírhattuk volna a FOR utasítás segítségével is, de a gépi kódú program is az előbbi szervezőmódot követi, így a két program közti analógia könnyebben felismerhető. Az assembler változat:

```
G/9 SEITE:1

033C 10 *=$33C
0400 20 KEP =$0400 ;KEP=1024
0800 30 SZIN =$0800 ;SZIN=55296
033C A0 00 40 LDY #0
033E A9 01 50 C1 LDA #1 ; A BETU KODJA
0340 99 00 04 60 STA KEP,Y
0343 A9 00 70 LDA #0 ;FEKETE SZIN KODJA
0345 99 00 D8 80 STA SZIN,Y
0348 C8 90 INY ;Y=Y+1
0349 98 95 TYA ;Y->AKKU
034A C9 A0 96 CMP #A0 ; HASONLITAS 160-AL
034C D0 F0 100 BNE C1
034E 60 110 RTS
034F 65535 .END

ZEILEN:14 SYMBOLE:3 FEHLER:0

C1 =033E KEP =0400 SZIN =0800
```

A program annyira egyszerű, hogy külön magyarázatot talán már nem igényel. A figyelmes Olvasó bizonyára észrevette, hogy ebben a példában a ciklusszervezés viszonylag egyszerű volt. Mivel a ciklusmagot csak 160-szor kellett megismételni, így az indexelést is könnyen oldottuk meg. Más a helyzet (de nem nehezebb), ha egy ciklusnak többször kell lefutnia, mint 256. Ekkor a címzést két indexregiszterrel és (ha szükséges) pl. indirekt indexelt címzésmóddal tudjuk megoldani. Tegyük fel, hogy a teljes képernyőt szeretnénk fekete A betűkkel teleírni. A BASIC programot

könnyen meg tudjuk írni, hiszen az az előzőnek egy módosított változata. Ennek elkészítését az Olvasóra bizzuk. Nézzük meg, hogyan oldható meg az a feladat assemblerben!

```
G/10 SEITE:1

033C 10 *=$33C
00FA 20 KEPA =$FA
00FB 30 KEPP =$FB
00FC 40 SZINA =$FC
00FD 50 SZINF =$FD
033C A9 00 70 LDA #$0400 ; A CIM ALSO BYTE-JA
033E 85 FA 80 STA KEPA
0340 A9 04 90 LDA #$0400 ; A CIM FELSO BYTE-JA
0342 85 FB 100 STA KEPP
0344 A9 00 110 LDA #$0800
0346 85 FC 120 STA SZINA
0348 A9 D8 130 LDA #$0800
034A 85 FD 140 STA SZINF
034C A9 00 145 LDA #0
034E A2 00 150 LDX #0
0350 A0 00 160 FOR1 LDY #0
0352 A9 01 200 FOR2 LDA #1 ; A BETU KODJA
0354 91 FA 210 STA (KEPA),Y ; INDIREKT INDEXELT CIM
0356 A9 00 220 LDA #0
0358 91 FC 230 STA (SZINA),Y ; INDIREKT INDEXELT CIM
035A C8 240 INY
035B D0 F5 250 BNE FOR2 ; BELSO CIKLUS VEGE PRINT
035D E6 FD 255 INC SZINF ; FELSO BYTE NOVELESE
035F E6 FB 256 INC KEPP ; FELSO BYTE NOVELESE
0361 E8 260 INX
0362 E0 04 270 CPX #04 ; KULSO CIKLUS VEGE PRINT
0364 D0 EA 280 BNE FOR1
0366 60 300 RTS
0367 65535 .END

ZEILEN:29 SYMBOLE:6 FEHLER:0

FOR1 =0350 FOR2 =0352 KEPA =00FA KEPP =00FB SZINA =00FC
SZINF =00FD
```

A programban indirekt indexelt címzésmódot alkalmaztunk. Mint azt már tudjuk, a nulláslapon a címek tárolásához egy kis helyre van szükségünk. Nem árt tudni azt sem, hogy a 251—254-es címek ún. szabad byte-ok, így átmeneti munkaterületként felhasználhatók. Ezt használtuk az előbbi programban mi is, a képernyő, ill. a szín RAM kezdőcímének tárolására.

6.7.6 Szubrutinok

Mint azt már a BASIC-ben is megszoktuk, ha egy programrészt többször kívánunk végrehajtani, akkor alprogramként (szubrutinként) írjuk meg. A főprogram és a szubrutin közti információcserét munkaváltozókkal kell megoldanunk. Írjuk meg azt a BASIC programot, amely a teljes karakterkészletet kiírja a képernyőre úgy, hogy először a teljes képernyőt teleírja A betűvel, majd B betűvel stb.

```
10 Kep=1024
20 szin=55296
30 Kar=1
40 gosub 100
50 if Kar>26 then stop
60 Kar=Kar+1
70 goto 40
100 rem Kiíró rutin
110 y=0
120 poke Kep+y,Kar
130 poke szin+y,0
140 y=y+1
150 if y=1000 then return
160 goto 120
```

A képernyőnek egy adott karakterrel való feltöltését a 100-as rutin végzi el. Nézzük meg a teljes programot assemblerben.

G/11

SEITE:1

```
0380      100      *=$380
00FA      110 KEPA  =$FA
00FB      120 KEPF  =$FB
00FC      130 SZINA  =$FC
00FD      140 SZINF  =$FD
0380 00    150 BETU  .BYTE 0
033C      160      *=$33C
033C A9 00 165      LDA #0 ; KEZDOERTEK BEALLITASA
033E 8D 80 03 166      STA BETU
0341 EE 80 03 170 C1    INC BETU
0344 20 4F 03 180      JSR IR
0347 AD 80 03 190      LDA BETU
034A C9 1A    200      CMP #26
034C D0 F3    210      BNE C1
034E 60      220      RTS
034F A9 00    230 IR    LDA #$0400< ; A CIM ALSO BYTE-JA
0351 85 FA    240      STA KEPA
0353 A9 04    250      LDA #$0400> ; A CIM FELSO BYTE-JA
0355 85 FB    260      STA KEPF
0357 A9 00    270      LDA #$0800<
0359 85 FC    280      STA SZINA
035B A9 D8    290      LDA #$0800>
035D 85 FD    300      STA SZINF
035F A2 00    310      LDX #0
0361 A9 00    320      LDA #0
0363 A0 00    330      LDY #0
0365 AD 80 03 340 FOR1  LDA BETU ; A BETU KODJA
0368 91 FA    350      STA (KEPA),Y ; INDIREKT INDEXELT CIM
036A A9 00    360      LDA #0
036C 91 FC    370      STA (SZINA),Y ; INDIREKT INDEXELT CIM
036E C8      380      INY
036F D0 F4    390      BNE FOR2 ; BELSO CIKLUS VEGE PRINT
0371 E6 FD    400      INC SZINF ; FELSO BYTE NOVELESE
0373 E6 FB    410      INC KEPF ; FELSO BYTE NOVELESE
0375 E8      420      INX
0376 E0 04    430      CPX #$04 ; KULSO CIKLUS VEGE PRINT
0378 D0 E9    440      BNE FOR1
037A 60      450      RTS
037B      65535     .END
```

ZEILEN:39 SYMBOLE:9 FEHLER:0

```
BETU  =0380 C1    =0341 FOR1  =0363 FOR2  =0365
IR     =034F KEPA  =00FA
KEPF   =00FB SZINA =00FC SZINF =00FD
```


Mint azt a listából is láthatjuk, az előző programot teljes egészében egy szubrutinnak tekintettük. Ennek hívását a 170—210-es sorszámú programrészlettel vezéreltük. Érdeemes összehasonlítani a BASIC és a gépi kódú program futtatási idejét!

6.7.7 INPUT-OUTPUT műveletek

Az input-output műveletek megvalósításához célszerű felhasználni az operációs rendszerben már meglévő rutinokat. A könnyebb felhasználás elő egítésére a gép készítői a tár utolsó lapján egy JUMP (ugrási) táblázatot helyeztek el (ez a KERNAL része), amelynek feladata, hogy végrehajtsa az operációs rendszer input-output rutinjait, valamint egyéb közhasznú segédprogramokat, majd a végrehajtás után visszahozza a vezérlést a felhasználói programnak. Bővebben erről a Data Becker — NO-VOTRADE kiadásában megjelent *A Commodore 64-es belső felépítése* c. könyvben olvashatunk. Könyvünk terjedelme ui. nem teszi lehetővé, hogy a KERNAL rutinokat részletesen ismertessük, ezért csak szemléltetésként veszünk egy példát.

A KERNAL rutinok használatához először be kell állítani a szükséges paramétereket, vagyis meg kell teremteni a kiválasztott rutin által megkívánt feltételeket. Ha ezeket elvégeztük, akkor a JSR utasítással hívhatjuk a kívánt rutint. Minden elérhető KERNAL rutin szubrutinszerűen épül fel, így mindig RTS utasítással végződik.

Módosítsuk az előző programot úgy, hogy az egyes karakterek kiírása után várjon egy tetszőleges billentyű leütésére! Egy karakter beolvasását a GETIN KERNAL rutinnal fogjuk elvégezni, amely a \$FFE4-es címen szerepel a táblázatban. A rutin egy karaktert olvas a billentyűzetről és annak ASCII kódját az akkumulátorba helyezi. Ha nem nyomtunk le egy billentyűt sem, akkor az akkumulátor tartalma 0 lesz (előkészítésként ide tehát nullát kell tölteni).

A várakozó rutin a következő:

```
GETIN      = $FFE4
.
.
.
WAR        LDA #0
WAIT       JSR GETIN
           CMP #0
           BEQ WAIT
           RTS
```

Ezt a rutint építettük be az előző programba, s így a teljes program listája a következő:

G/12

SEITE:1

0390	100	*= \$390
00FA	110 KEPA	= \$FA
00FB	120 KEPF	= \$FB
00FC	130 SZINA	= \$FC
00FD	140 SZINF	= \$FD
FFE4	145 GETIN	= \$FFE4
0390 00	150 BETU	.BYTE 0
033C	160	*= \$33C
033C A9 00	165	LDA #0 ; KEZDOERTEK BEALLITASA
033E 8D 90 03	166	STA BETU
0341 EE 90 03	170 C1	INC BETU
0344 20 52 03	180	JSR IR
0347 20 7E 03	185	JSR VAR
034A AD 90 03	190	LDA BETU
034D C9 1A	200	CMP #26
034F D0 F0	210	BNE C1
0351 60	220	RTS
0352 A9 00	230 IR	LDA #\$0400< ; A CIM ALSO BYTE-JA
0354 85 FA	240	STA KEPA
0356 A9 04	250	LDA #\$0400> ; A CIM FELSO BYTE-JA
0358 85 FB	260	STA KEPF
035A A9 00	270	LDA #\$0800<
035C 85 FC	280	STA SZINA
035E A9 08	290	LDA #\$0800>
0360 85 FD	300	STA SZINF
0362 A2 00	310	LDX #0
0364 A9 00	320	LDA #0
0366 A0 00	330 FOR1	LDY #0
0368 AD 90 03	340 FOR2	LDA BETU ; A BETU KODJA
036B 91 FA	350	STA (KEPA),Y ; INDIREKT INDEXELT CIM
036D A9 00	360	LDA #0
036F 91 FC	370	STA (SZINA),Y ; INDIREKT INDEXELT CIM
0371 C8	380	INY
0372 D0 F4	390	BNE FOR2 ; BELSO CIKLUS VEGE PRINT
0374 E6 FD	400	INC SZINF ; FELSO BYTE NOVELESE
0376 E6 FB	410	INC KEPF ; FELSO BYTE NOVELESE
0378 E8	420	INX
0379 E0 04	430	CPX #\$04 ; KULSO CIKLUS VEGE PRINT
037B D0 E9	440	BNE FOR1
037D 60	450	RTS
037E A9 00	500 VAR	LDA #0
0380 20 E4 FF	510 WAIT	JSR GETIN
0383 C9 00	520	CMP #0
0385 F0 F9	530	BEQ WAIT
0387 60	540	RTS
0388	65535	.END

ZEILEN:46 SYMBOLE:12 FEHLER:0

BETU	=0390	C1	=0341	FOR1	=0366	FOR2	=0368
GETIN	=FFE4	IR	=0352				
KEPA	=00FA	KEPF	=00FB	SZINA	=00FC	SZINF	=00FD
VAR	=037E	WAIT	=0380				

Ezzel a példával befejeztük a gépi kódú programozás ismertetését. Reméljük, hogy azok számára, akik eddig csak BASIC-ben programoztak, sikerült bebizonyítani, hogy a gépi kódú programozás sem olyan ördögös dolog, s némi analógia is felfedezhető a két programozási stílus között.

FÜGGELÉK

HELP + ÖSSZEFOGLALÓ

#A: APPEND

Szintaxis: #A "NÉV", D (ahol a D = az egység száma)

Ha a D hiányzik, alapértelmezés: kazettás egység, ha akkor a név hiányzik, a következő programot olvassa.

Hatása: a beolvasott program a tárban levő program *mögé* kerül. A betöltött program sorszámozása lehet kisebb vagy azonos a tárban levővel. A programok hossza lehet különböző.

#H: HELP

#H Programmegszakításkor ezzel a paranccsal az utoljára végrehajtott program-sort kiírathatjuk.

Ha a megszakítás hiba miatt történt, a hibás rész riverz módban jelenik meg.

A parancsot rögtön a megszakítás után kell kiadni, amennyiben szükséges!

#F: FIND

Szintaxis: #F PARANCS (Minden sort kiír, amelyben ez a parancs előfordul)

#F VÁLTOZÓ (Minden sort kiír, amelyben ez a változó előfordul)

#F "SZÖVEG" (Minden sort kiír, amelyben ez a szöveg előfordul)

Ha a változót csak egy betűvel adjuk meg, minden ezzel kezdődő azonosítót kiír.

Ha szöveget üresen keresünk, #F"", minden szöveget tartalmazó sort kilistáz.

Parancs keresésénél rövidíthetjük a nevet, OPEN helyett OP (ahol a P billentyűt a SHIFT billentyűvel együtt nyomtuk le).

A listázást a CTRL lassítja, a SHIFT megállítja.

STOP — kilépés

CMD-vel nyomtatóra.

#D: DELETE

Szintaxis: #D 100 A 100-as sort törli.

#D -100 A 100-as sorig mindent töröl (a 100-ast is).

#D 100- A 100-as sortól mindent töröl (a 100-ast is).

#D 100-200 A 100-astól a 200-asig töröl minden sort.

#D Hibajelzést ad.

A programhoz fűzött gépi rutinok megmaradnak.

DOS TÁMOGATÁS I

Lemezparancsok rövidítése

% *Betöltés* (nem a BASIC által megadott kezdőcímmel tölti a programot, hanem a lemezen megadottra)

↑ *Betöltés* BASIC-területre + RUN automatikusan

→ *SAVE*

< *Verify*

DOS TÁMOGATÁS

15-ös csatorna parancsai (parancscsatorna parancsok)

@N0 : NÉV, ID NEW parancs lemezre

@I A lemez inicializálása

@S0: NÉV Töröl a lemeztől egy programot

@B—12: 2 0 5 2 Blokk olvasása a 2-es csatornán a 0-s meghajtó 5-ös sáv 2-es szektorból

\$: Tartalomjegyzék kiírása a képernyőterületre.

(A BASIC program a tárban marad)

CMD: Parancs segítségével lehet kiírni

#V: VARIABLE DUMP

Szintaxis: #V

Programfutás után a skalárváltozók pillanatnyi értékét kilistázza. A változók sorrendje megegyezik az előfordulás sorrendjével. A listát a CTRL gombbal lehet lassítani, a SHIFT-tel leállítani, STOP-pal kiépni. CMD-vel nyomtatóra.

#M: MATRIX DUMP

#M Ugyanez, mint #V tömbökre.

!#: ÁTSZÁMOLÁS hexadecimálisról decimálisra

!#

!# HHHH

Visszaadja a beírt hexadecimális számot decimálisan.

!\$: ÁTSZÁMOLÁS decimálisról hexadecimálisra

!\$ decimális szám (0—65535)

#B: BASIC RENEW

Háromféle felhasználása:

1. NEW-vel törölt program aktivizálása.

2. Egy BASIC programhoz fűzött gépi kódú részt, vagy OVERLAY miatt tévesen odaírt programrészt törölhetjük. (Átállítja a Programvég-mutatót.)

3. Egy BASIC program betöltése után, ha nincs kurzor és a kurzor csak a STOP-RESTOR-ra jön elő, akkor ezzel a paranccsal a programmutatót újra beállíthatjuk.

A parancsot nem szabad használni, ha nincs BASIC program a tárban, egyébként ui. keletkezhetnek a parancs hatására definiálatlan tartalmú programsorok.

#G: GENLINE

Szintaxis: #G A,B A = kezdősorszám

B = végsorszám

A, B nélkül 100, 10

Minden RETURN-re ad egy új sorszámot.

#T: TRACE

#T hatására a program soronként hajtódik végre, és minden sor után van egy kis szünet a végrehajtásban. Ha futás közben a SHIFT-et megnyomjuk, jelentősen lelassul. A RUN, GOTO, GOSUB vagy CONT parancsra a végrehajtásra kerülő sor száma megjelenik a képernyő tetején. Egyszerre mindig 6 sorszámot láthatunk. A következő végrehajtandó utasítás tartalma a 2. és a 3. képernyő-sorban jelenik meg. Ha közben ide INPUT-ot kér a program, a kurzort a 4. sorba visszük és beírhatjuk az adatokat.

STOP-pal leállítható, CONT-tal folytatható a program.

Kikapcsolás #E = END TRACE.

A 6 sorszám a \$0122—\$0130-os címeken tárolódik.

Nem szabad felülrni!

#S: SINGLE STEP (egy lépés)

Soronként hajtjuk végre vele a programot. A parancs hatására a feldolgozás üteme olyan, hogy két utasítás végrehajtása közben megnyomjuk valamelyik SHIFT gombot. Ha megnyomjuk, a feldolgozás lelassul. Itt is megjelenik az utolsó 6 sorszám.

INPUT-ra ugyanaz vonatkozik, mint a TRACE-nél.

Kikapcsolás #E-vel.

#R: RENUMBER — a tárolóban levő program átsorszámozása

#R A,B A = kezdő sorszám; B = növekmény

#R A és B nélkül 100, 10

(FILE megnyitás és CMD

(d = egység (4—31)

Ha nincs d, akkor automatikusan d=4.

) FILE lezárás

) Befejezi a CMD-t és lezárja a megnyitott csatornát.

£ EGYSÉGSZÁM megváltoztatása a DOS-nak

£d (d: 4—31)

#K: KILL

#K

A HELP C 64-es PLUS kikapcsolása.

#C: COMPAKTOR

#C d d = a tömörítés foka (1—240)

#C d = 240

Hatására minden fölös szóköz és REM törlődik. A sorokat a d-ben megadott hosszra tömöríti. Végrehajtása előtt UNDEF'D STATEMENT hibát keres a programban, ha ilyen van, nem tömörít.

#C után a sorszámozás 1—1, futása gyorsabb.

#U: UNDEF'D STATEMENT vizsgálat

#U Megkeresi a „definiálatlan utasítás” hibát.

*: SZABAD TÁRTERÜLET

* a PRINT FRE (0)-t helyettesíti.

KÉPERNYŐKÓDOK

1.Készlet 2.Készlet Poke

@		0
A	a	1
B	b	2
C	c	3
D	d	4
E	e	5
F	f	6
G	g	7
H	h	8
I	i	9
J	j	10
K	k	11
L	l	12
M	m	13
N	n	14
O	o	15
P	p	16
Q	q	17
R	r	18
S	s	19
T	t	20
U	u	21
V	v	22
W	w	23
X	x	24
Y	y	25
Z	z	26

1.Készlet 2.Készlet Poke

[	27
£	28
]	29
↑	30
←	31
SPACE	32
!	33
"	34
#	35
\$	36
%	37
&	38
,	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47
0	48
1	49
2	50
3	51
4	52
5	53

1.Készlet 2.Készlet Poke

6		54
7		55
8		56
9		57
:		58
;		59
<		60
=		61
>		62
?		63
☐		64
☐	A	65
☐	B	66
☐	C	67
☐	D	68
☐	E	69
☐	F	70
☐	G	71
☐	H	72
☐	I	73
☐	J	74
☐	K	75
☐	L	76
☐	M	77
☐	N	78
☐	O	79
☐	P	80

1.Készlet 2.Készlet Poke

	Q	81
	R	82
	S	83
	T	84
	U	85
	V	86
	W	87
	X	88
	Y	89
	Z	90
		91
		92
		93
		94
		95
	SPACE	96

1.Készlet 2.Készlet Poke

		97
		98
		99
		100
		101
		102
		103
		104
		105
		106
		107
		108
		109
		110
		111
		112

1.Készlet 2.Készlet Poke

		113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125
		126
		127

ASCII ÉS CHR\$ KÓDOK

A táblázat megmutatja egyrészt, hogy milyen karakterek jelennek meg a PRINTCHR\$ (X) utasítás végrehajtásakor, ahol X a CHR\$ oszlop valamely értéke; másrészt a PRINTASC("x") utasítás végrehajtásakor minden gépelhető "x" karakternek megfelelően a CHR\$ oszlop kódértékeit adja.

A funkcionális billentyűk utasításbeli felhasználásakor is hasznos segítséget nyújt.

Karakter	CHR\$	Karakter	CHR\$	Karakter	CHR\$	Karakter	CHR\$
	0		23	.	46	E	69
	1		24	/	47	F	70
	2		25	0	48	G	71
	3		26	1	49	H	72
	4		27	2	50	I	73
	5		28	3	51	J	74
	6		29	4	52	K	75
	7		30	5	53	L	76
	8		31	6	54	M	77
tilt	9		32	7	55	N	78
engedélyez	10		33	8	56	O	79
	11	!	34	9	57	P	80
	12	"	35	:	58	Q	81
	13	#	36	;	59	R	82
	14	\$	37	<	60	S	83
	15	%	38	=	61	T	84
	16	&	39	>	62	U	85
	17	.	40	?	63	V	86
	18	(	41	@	64	W	87
	19	)	42	A	65	X	88
	20	*	43	B	66	Y	89
	21	+	44	C	67	Z	90
	22	,	45	D	68	[	91
		-					

Karakter	CHR\$	Karakter	CHR\$	Karakter	CHR\$	Karakter	CHR\$
£	92		124		156		184
]	93		125		157		185
↑	94		126		158		186
↑	95		127		159		187
	96		128		160		188
	97	narancs	129		161		189
	98		130		162		190
	99		131		163		191
	100		132		164		
	101	f1	133		165		
	102	f3	134		166		
	103	f5	135		167		
	104	f7	136		168		
	105	f2	137		169		
	106	f4	138		170		
	107	f6	139		171		
	108	f8	140		172		
	109		141		173		
	110		142		174		
	111		143		175		
	112		144		176		
	113		145		177		
	114		146		178		
	115		147		179		
	116		148		180		
	117	barna	149		181		
	118	vil.piros	150		182		
	119	szürke 1	151		183		
	120	szürke 2	152				
	121	vil.zöld	153				
	122	vil.kék	154				
	123	szürke 3	155				

192-233 kódok megegyeznek a
96-127 kódokkal 224-254 kódok
megegyeznek a 160-190 kódokkal
255 kód megegyezik a 126 kóddal

BASIC KULCSSZAVAK RÖVIDÍTÉSE

A Commodore 64-es BASIC változata lehetővé teszi a kulcsszavak rövidítését. Így pl. a PRINT helyett röviden a kérdőjel használható. A többi rövidíthető kulcsszót az első, vagy az első és a második betű, valamint a **SHIFT** billentyűvel leütött következő betű együttesen rövidíti, pl.:

ABS helyett A **SHIFT** B
CLOSE helyett CL **SHIFT** O

használható a program írásához.

LIST parancs végrehajtása után a teljes kulcsszó íródik ki.

Utasi- tás	Rövidi- tés	Kép- ernyő	Utasi- tás	Rövidi- tés	Kép- ernyő
FRE	F SHIFT R	F	ABS	A SHIFT B	A
GET	G SHIFT E	G	AND	A SHIFT N	A
GET#	NONE	GET#	ASC	A SHIFT S	A
GOSUB	GO SHIFT S	GO	ATN	A SHIFT T	A
GOTO	G SHIFT	G	CHR\$	C SHIFT H	C
IF	NONE	IF	CLOSE	CL SHIFT O	CL
INPUT	NONE	INPUT	CLR	C SHIFT L	C
INPUT#	I SHIFT N	I	CMD	C SHIFT M	C
INT	NONE	INT	CONT	C SHIFT O	C
LET\$	LE SHIFT F	LE	COS	NONE	COS
LEN	NONE	LEN	DATA	D SHIFT A	D
LET	L SHIFT E	L	DEF	D SHIFT E	D
LIST	L SHIFT I	L	DIM	D SHIFT I	D
LOAD	L SHIFT O	L	END	E SHIFT N	E
LOG	NONE	LOG	EXP	E SHIFT X	E
MID\$	M SHIFT I	M	FN	NONE	FN
NEW	NONE	NEW	FOR	F SHIFT O	F

Utasi- tás	Rövidi- tés	Kép- ernyő	Utasi- tás	Rövidi- tés	Kép- ernyő
NEXT	N SHIFT E	N ☐	SGN	S SHIFT G	S ☐
NOT	N SHIFT Ø	N ☐	SIN	S SHIFT I	S ☐
ON	NONE	ON	SPC(	S SHIFT P	S ☐
OPEN	O SHIFT P	O ☐	SQR	S SHIFT Q	S ☒
OR	NONE	OR	STATUS	ST	ST
PEEK	P SHIFT E	P ☐	STEP	ST SHIFT E	ST ☐
POKE	P SHIFT O	P ☐	STOP	S SHIFT T	S ☐
POS	NONE	POS	STR\$	ST SHIFT R	ST ☐
PRINT	?	?	SYS	S SHIFT Y	S ☐
PRINT#	P SHIFT R	P ☐	TAB(	T SHIFT A	T ☒
READ	R SHIFT E	R ☐	TAN	NONE	TAN
REM	NONE	REM	THEN	T SHIFT H	T ☐
RESTORE	RE SHIFT S	RE ☒	TIME	TI	TI
RETURN	RE SHIFT T	RE ☐	TIME\$	TI\$	TI\$
RIGHT\$	R SHIFT I	R ☐	USR	U SHIFT S	U ☒
RND	R SHIFT N	R ☒	VAL	V SHIFT A	V ☒
RUN	R SHIFT U	R ☒	VERIFY	V SHIFT E	V ☐
SAVE	S SHIFT A	S ☒	WAIT	W SHIFT A	W ☒

BASIC HIBAÜZENETEK

BAD DATA	→ Hibás adat — pl. egy megnyitott adatállományból füzér érkezett, de a program számadatát vár.
BAD SUBSCRIPT	→ Hibás index — a program egy tömb olyan elemére hivatkozik, amely a DIM utasításban megadott határon kívül esik.
CANT CONTINUE	→ Nem folytatható — a CONT parancsot nem tudja végrehajtani, mert a program még nem futott, vagy hibás, vagy közben szerkesztettük valamelyik sort.
DEVICE NOT PRESENT	→ Nem elérhető periféria — a szükséges bemeneti vagy kimeneti készülék nincs csatlakoztatva, így az OPEN CLOSE, CMD, INPUT #, GET#, PRINT# parancsot nem tudja végrehajtani.
DIVISION BY ZERO	→ Nullával való osztás — matematikailag nem értelmezhető.
EXTRA IGNORED	→ Felesleges visszautasítva — az INPUT utasítás után túl sok adatot gépeltünk be, csak az első néhány került elfogadásra.
FILE NOT FOUND	→ Nem találta az adatállományt — keresés közben "END OF TAPE" (szalag vége) jelzést talált, ill. lemez esetén nincs ilyen nevű program.
FILE NOT OPEN	→ Nincs megnyitva az adatállomány — a CLOSE, CMD, INPUT#, GET#, PRINT# utasításban megjelölt adatállományt még nem nyitottuk meg.
FILE OPEN	→ Adatállomány megnyitva — kísérlet történt egy, már megnyitott adatállomány újbóli megnyitására.
FORMULA TOO COMPLEX	→ Túlzottan összetett formula — a kiértékelendő füzér kifejezést legalább két részre kell bontani, hogy a rendszer dolgozni tudjon vele.
ILLEGAL DIRECT	→ Illegális parancs — az utasítás csak programon belül használható, parancsként nem.
ILLEGAL QUANTITY	→ Illegális mennyiség — az utasítás vagy függvény argumentumaként megadott szám kívül esik a megengedett tartományon.

LOAD	→ Töltés — a szalagon levő programmal probléma van.
NEXT WITHOUT FOR	→ FOR nélküli NEXT — hibásan egymásba ágyazott ciklusok vagy a FOR és NEXT utasításban megadott változónév nem azonos.
NOT INPUT FILE	→ Nem bemeneti adatállomány — kimenetként megjelölt adatállományt akartunk beolvasni INPUT# vagy GET# utasításokkal.
NOT OUTPUT FILE	→ Nem kimeneti adatállomány — bemenetként megjelölt adatállományba akartunk a PRINT# utasítással írni.
OUT OF DATA	→ Kifutott az adatból — a READ utasítás nem hajtható végre, mert a DATA utasításokban elhelyezett valamennyi adatot már beolvasta.
OUT OF MEMORY	→ Betelt a tár — nem áll rendelkezésre több tárhely a program, ill. a változók részére, vagy túl sok FOR-NEXT, ill. GOSUB-RETURN utasítást ágyaztunk egymásba.
OVERFLOW	→ Túlcsordulás — a számítás eredménye nagyobb, mint a megengedett legnagyobb szám ($1.70141884E+38$).
REDIM'D ARRAY	→ Újradienzionált tömb — egy tömböt csak egyszer lehet dimenzionálni, ha egy tömbváltozót már használtunk, automatikusan végrehajtható a DIM művelet (a tömb elemszáma 11 lesz).
REDO FROM START	→ Újrakezdés — az INPUT utasításban számadat helyett karakter adatot olvastunk be, a helyes bemenő érték begépelése esetén a program automatikusan folytatódik.
RETURN WITHOUT GOSUB	→ GOSUB nélküli RETURN — a program RETURN utasítást talált, az ezt megelőző GOSUB hívás nélkül.
SYNTAX	→ Szintaktikai hiba — a Commodore 64-es által az utasítás felismerhetetlen. (Hiányzó vagy felesleges zárójelek. Elírt kulcsszavak stb.)
TYPE MISMATCH	→ Típuskeveredés — füzér helyett számot használtunk, ill. fordítva.
UNDEF'D FUNCTION	→ Definiálatlan függvény — egy felhasználó által definiált függvényre hivatkoztunk, amit azonban még nem adtunk meg a DEF FN utasítással.

UNDEF'D STATEMENT	→ Definiálatlan utasítás — a GOTO, GOSUB, RUN stb. utasítás egy nem létező sorszámmal utal.
VERIFY	→ Összehasonlítás — a szalagra vagy lemezre kiírt program nem egyezik a tárból olvasottal.

A LEMEZEGYSÉG HIBAÜZENETEI

- 00, OK, 00, 00
Az utolsó lemezművelet hibamentesen zajlott le.
- 01, FILES SCRATCHED, XX, 00
Ez egy SCRATCH utasítást követő visszajelzés. Az XX szám adja meg a törölt file-ok számát.
- 20, READ ERROR, TT, SS
Olvasási hiba. A TT és az SS arra a sávra és szektorra vonatkozik, amelynél a hiba fellépett.
- 21, READ ERROR, TT, SS
Ez is olvasási hiba.
- 22, READ ERROR, TT, SS
Egy adatblokk fejlécében ellenőrzőösszeg (kontrollsumma) hiba van.
- 23, READ ERROR, TT, SS
Egy olyan adatblokkot olvastunk a DOS-pufferbe, amelyben ellenőrzőösszeg hiba van. Egy vagy több adatbyte hibás.
- 24, READ ERROR, TT, SS
Hibás file-okat próbálunk beolvasni.
- 25, WRITE ERROR, TT, SS
Írásellenőrzés hiba.
- 26, WRITE PROTECT ON, TT, SS
Írásvédelem. A lemezen rajta van az írásvédelmi tapasz.
- 27, READ ERROR, TT, SS
Olvasási hiba a fejlécben (headerben).
- 28, WRITE ERROR, TT, SS
Íráshiba (hosszú adatblokk).
- 29, DISK IN MISMATCH, TT, SS
Lemez ID-hiba. A lemez ID-je nem azonos a DOS-ban tárolt ID-vel.
- 30, SYNTAX ERROR, 00, 00
Szintaktikus hiba. A DOS nem tudja értelmezni a parancscsatornára kapott utasítást.
- 31, SYNTAX ERROR, 00, 00
Szintaktikus hiba (nem létező utasítás).
- 32, SYNTAX ERROR, 00, 00
Szintaktikus hiba (hosszú sor). A kapott urasítás 40 karakternél hosszabb.
- 33, SYNTAX ERROR, 00, 00
Szintaktikus hiba (nem létező file-név).
- 34, SYNTAX ERROR, 00, 00
Szintaktikus hiba (nincs megadva a file).
- 39, FILE NOT FOUND, 00, 00
Szintaktikus hiba (nem létező file).
- 50, RECORD NOT PRESENT, 00, 00
A rekord nincs jelen.
- 51, OVERFLOW IN RECORD, 00, 00
Túlsordulás a rekordban.
- 52, FILE TOO LARGE, 00, 00
A file túl nagy.

- 60, WRITE FILE OPEN, 00, 00
A file írásra van nyitva.
- 61, FILE NOT OPEN, 00, 00
A file nincs megnyitva.
- 62, FILE NOT FOUND, 00, 00
Nincs meg a file.
- 63, FILE EXISTS, 00, 00
A file létezik. A létrehozott file neve a lemezen egyszer már megtalálható.
- 64, FILETYPE MISMATCH, 00, 00
File-típus hiba.
- 65, NO BLOCK, TT, SS
A lefoglalandó blokk már nem szabad. A DOS ebben az esetben automatikusan egy, az első szabad blokkhoz tartozó sáv/szektor értékét írja ki. Ha már nincs szabad blokk, akkor két 0-t ír ki.
- 66, ILLEGAL TRACK OR SECTOR, TT, SS
Illegális sáv vagy szektor.
- 67, ILLEGAL TRACK OR SECTOR, TT, SS
Egy file sáv/szektor láncolása egy nem létező sávra vagy szektorra mutat.
- 70, NO CHANNEL, 00, 00
Nincs csatorna.
- 71, DIR ERROR, TT, SS
Tartalomjegyzék-hiba.
- 72, DISK FULL, 00, 00
A lemezen már 3-nál kevesebb szabad blokk van, vagy a rendszer elérte a tartalomjegyzék-bejegyzések maximális számát (A VC—1541-nél: 144).
- 73, CBM DOS V2.6 1541, 00, 00
Ez az üzenet a VC—1541-es készülék bekapcsolási üzenete.
- 74, DRIVE NOT READY, 00, 00
Ez a hibaüzenet akkor jelentkezik, ha lemezműveletet akarunk végezni anélkül, hogy lemez lenne a meghajtóegységben vagy az egység be lenne kapcsolva.

A 6510-es mikroprocesszor gépi kódú utasításai

Az egyes utasítások rövid leírásánál a következő jelöléseket alkalmazzuk:

- A akkumulátor;
- X,Y indexregiszter;
- M tár (memória);
- Op operandus (egy utasításban).

Az utasítások mellett mindig megjelöljük, hogy az állapotregiszter melyik bitje módosul (a bitek jelölését l. a 6.4 alfejezetben).

ADC

A tár és az akkumulátor tartalmát adja össze. Az eredmény az akkumulátorban lesz. Ha volt átvitel, akkor az állapotregiszterben a C bitet 1-re állítja.

Címzés mód	Assembler	Műveleti kód
Közvetlen	ADC # Op	69
Nulláslap	ADC Op	65
Nulláslap, X	ADC Op, X	75
Abszolút	ADC Op	60
Abszolút, X	ADC Op, X	70
Abszolút, Y	ADC Op, Y	79
(Indirekt), X	ADC (Op), X	61
(Indirekt), Y	ADC (Op), Y	71

AND

AND művelet az akkumulátor és a rekesz tartalma között. A logikai művelet eredménye az A-ba kerül.

Címzés mód	Assembler	Műveleti kód
Közvetlen	AND # Op	29
Nulláslap	AND Op	25
Nulláslap, X	AND Op, X	35
Abszolút	AND Op	2D
Abszolút, X	AND Op, X	3D
Abszolút, Y	AND Op, Y	39
(Indirekt), X	AND (Op), X	21
(Indirekt), Y	AND (Op), Y	31

ASL

Balra tolás 1 bittel (akkumulátorban vagy egy rekeszben). A balra kilépő bit a C kapcsolóba kerül.

Címzés mód	Assembler	Műveleti kód
Akkumulátor	ASL A	0A
Nulláslap	ASL Op	06
Nulláslap, X	ASL Op, X	16
Abszolút	ASL Op	0E
Abszolút, X	ASL Op, X	1E

BCC

Elágazás, ha nincs átvitel ($C=0$)

Címzés mód	Assembler	Műveleti kód
Relatív	BCC Op	90

BCS

Elágazás, ha volt átvitel ($C=1$)

Címzés mód	Assembler	Műveleti kód
Relatív	BCS Op	B0

BEQ

Elágazás, ha az eredmény nulla ($Z=1$).

Címzés mód	Assembler	Műveleti kód
Relatív	BEQ	F0

BIT

Ez az utasítás csak a kapcsolókat állítja, miközben a regiszterek tartalma nem változik. A processzor logikai AND műveletet végez az akkumulátor és az adott tárcím tartalma között. Ha az eredmény 0, akkor $Z=1$, egyébként $Z=0$ lesz. Ugyanakkor a tárcím 6. bitjét a V kapcsolóba, a 7. bitjét pedig az N kapcsolóba tölti.

Címzés mód	Assembler	Műveleti kód
0-ás lap	BIT Op	24
Abszolút	BIT Op	2C

BMI

Elágazás, ha az eredmény negatív ($N=1$).

Címzés mód	Assembler	Műveleti kód
Relatív	BMI Op	30

BNE

Elágazás, ha az eredmény nem nulla ($Z=0$).

Címzés mód	Assembler	Műveleti kód
Relatív	BNE Op	D0

BPL

Elágazás, ha az eredmény pozitív vagy nulla ($N=0$).

Címzés mód	Assembler	Műveleti kód
Relatív	BPL Op	10

BRK

Feltétel nélküli programmegszakítás, az utasításszámláló és az állapotregiszter tartalmát a verembe helyezi. Majd elágazás következik a \$FFFE és a \$FFFF címben levő rutinra.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	BRK	00

BVC

Elágazás, ha nem volt túlsordulás ($V=0$).

Címzés mód	Assembler	Műveleti kód
Relatív	BVC Op	50

BVS

Elágazás, ha volt túlsordulás ($V=1$).

Címzés mód	Assembler	Műveleti kód
Relatív	BVS Op	70

CLC

Átviteli kapcsoló törlése ($0 \rightarrow C$).

Címzés mód	Assembler	Műveleti kód
Cím nélküli	CLC	18

CLD

Decimális mód törlése ($0 \rightarrow D$).

Címzés mód	Assembler	Műveleti kód
Cím nélküli	CLD	D8

CLI

A megszakítást letiltó bit törlése ($0 \rightarrow I$).

Címzés mód	Assembler	Műveleti kód
Cím nélküli	CLI	58

CLV

Túlsordulási kapcsoló törlése ($0 \rightarrow V$).

Címzés mód	Assembler	Műveleti kód
Cím nélküli	CLV	B8

CMP

Az akkumulátor és a rekesz tartalmának összehasonlítása úgy, hogy a megadott tárcím tartalmát kivonja az akkumulátor tartalmából.

Ha $A-M > 0$, akkor $C=1$, egyébként 0 ($Z=0, N=0$);
ha $A=M$, akkor $Z=1$, egyébként 0 ($C=1, N=0$);
ha $A-M > \$7F$, akkor $N=1$, egyébként 0 ($C=0, Z=0$).

Címzés mód	Assembler	Műveleti kód
Közvetlen	CMP #Op	C9
Nulláslap	CMP Op	C5
Nulláslap, X	CMP Op, X	D5
Abszolút	CMP Op	CD
Abszolút, X	CMP Op, X	DD
Abszolút, Y	CMP Op, Y	D9
(Indirekt), X	CMP (Op), X	C1
(Indirekt), Y	CMP (Op), Y	D1

CPX

A tárcím és az X index összehasonlítása. Hasonló módon, mint CMP-nél, csak az akkumulátor helyett az X regiszter van megadva.

Címzés mód	Assembler	Műveleti kód
Közvetlen	CPX #Op	E0
Nulláslap	CPX Op	E4
Abszolút	CPX Op	EC

CPY

A tárcím és az Y index összehasonlítása. Hasonló módon, mint CMP-nél, csak az akkumulátor helyett az Y regiszter van megadva.

Címzés mód	Assembler	Műveleti kód
Közvetlen	CPY #Op	C0
Nulláslap	CPY Op	C4
Abszolút	CPY Op	CC

DEC

A tárcím tartalmának csökkentése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Nulláslap	DEC Op	C6
Nulláslap, X	DEC Op, X	D6
Abszolút	DEC Op	CE
Abszolút, X	DEC Op, X	DE

DEX

A X index tartalmának csökkentése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	DEX	CA

DEY

Az Y index tartalmának csökkentése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	DEX	88

EOR

„Kizáró vagy” logikai művelet a tárcím és akkumulátor között.

Címzés mód	Assembler	Műveleti kód
Közvetlen	EOR #Op	49
Nulláslap	EOR Op	45
Nulláslap, X	EOR Op, X	55
Abszolút	EOR Op	4D
Abszolút, X	EOR, Op, X	5D
Abszolút, Y	EOR Op, Y	59
(Indirekt), X	EOR (Op), X	41
(Indirekt), Y	EOR (Op), Y	51

INC

A rekesz tartalmának növelése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Nulláslap	INC Op	E6
Nulláslap, X	INC Op, X	F6
Abszolút	INC Op	EE
Abszolút, X	INC Op, X	FE

INX

Az X index regiszter tartalmának növelése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	INX	E8

INY

Az Y index regiszter tartalmának növelése 1-gyel.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	INY	C8

JMP

Ugrás a megadott címre.

Címzés mód	Assembler	Műveleti kód
Abszolút	JMP Op	4C
Indirekt	JMP (Op)	6C

JSR

Szubrutin hívása.

Címzés mód	Assembler	Műveleti kód
Abszolút	JSR Op	20

LDA

A rekesz tartalmának az akkumulátorba töltése.

Címzés mód	Assembler	Műveleti kód
Közvetlen	LDA # Op	A9
Nulláslap	LDA Op	A5
Nulláslap, X	LDA Op, X	B5
Abszolút	LDA Op	AD
Abszolút, X	LDA Op, X	BD
Abszolút, Y	LDA Op, Y	B9
(Indirekt), X	LDA (Op), X	A1
(Indirekt), Y	LDA (Op), Y	B1

LDX

A rekesz tartalmának az X indexregiszterbe töltése.

Címzés mód	Assembler	Műveleti kód
Közvetlen	LDX # Op	A2
Nulláslap	LDX Op	A6
Nulláslap, Y	LDX Op, Y	B6
Abszolút	LDX Op	AE
Abszolút, Y	LDX Op, Y	BE

LDY

A rekesz tartalmának Y indexregiszterbe töltése.

Címzés mód	Assembler	Műveleti kód
Közvetlen	LDY # Op	A0
Nulláslap	LDY Op	A4
Nulláslap, X	LDY Op, X	B4
Abszolút	LDY Op	AC
Abszolút, X	LDY Op, X	BC

LSR

Eltolás jobbra 1 bittel (rekeszben vagy akkumulátorban).
A jobbra kilépő bit a C kapcsolóba kerül.

Címzés mód	Assembler	Műveleti kód
Akkumulátor	LSR A	4A
Nulláslap	LSR Op	46
Nulláslap, X	LSR Op, X	56
Abszolút	LSR Op	4E
Abszolút, X	LSR Op, X	5E

NOP

Üres művelet, a processzor ennél az utasításnál nem csinál semmit.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	NOP	EA

ORA

„Vagy” logikai művelet az akkumulátor és egy rekesz tartalmával.

Címzés mód	Assembler	Műveleti kód
Közvetlen	ORA # Op	09
Nulláslap	ORA Op	05
Nulláslap, X	ORA Op, X	15
Abszolút	ORA Op	0D
Abszolút, X	ORA Op, X	1D
Abszolút, Y	ORA Op, Y	19
(Indirekt), X	ORA (Op), X	01
(Indirekt), Y	ORA (Op), Y	11

PHA

Az akkumulátor tartalmának a verembe helyezése.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	PHA	48

PHP

A processzor állapotregiszterének tartalmát a verembe tölti.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	PHP	08

PLA

A verem tartalmának áttöltése az akkumulátorba.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	PLA	68

PLP

A verem tartalmának visszatöltése a processzor állapotregiszterébe.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	PLP	28

ROL

Ciklikuseltolás balra 1 bittel (rekesz vagy akkumulátor). A balra kilépő bit a C kapcsolóba kerül.

Címzés mód	Assembler	Műveleti kód
Akkumulátor	ROL A	2A
Nulláslap	ROL Op	26
Nulláslap, X	ROL Op, X	36
Abszolút	ROL Op	2E
Abszolút, X	ROL Op, X	3E

ROR

Ciklikuseltolás 1 bittel jobbra (rekesz vagy akkumulátor). A jobbra kilépő bit a C kapcsolóba kerül.

Címzés mód	Assembler	Műveleti kód
Akkumulátor	ROR A	6A
Nulláslap	ROR Op	66
Nulláslap, X	ROR Op, X	76
Abszolút	ROR Op	6E
Abszolút, X	ROR Op, X	7E

RTI

Visszatérés megszakításból.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	RTI	40

RTS

Visszatérés a szubrutinból.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	RTS	60

SBC

A rekesz tartalmának kivonása az akkumulátorból áthozattal.

Címzés mód	Assembler	Műveleti kód
Közvetlen	SBC # Op	E9
Nulláslap	SBC Op	E5
Nulláslap, X	SBC Op, X	F5
Abszolút	SBC Op	ED
Abszolút, X	SBC Op, X	FD
Abszolút, Y	SBC Op, Y	F9
(Indirekt), X	SBC (Op), X	E1
(Indirekt), Y	SBC (Op), Y	F1

SEC

Átviteli kapcsoló beállítása

Címzés mód	Assembler	Műveleti kód
Cím nélküli	SEC	38

SED

Decimális üzemmód beállítása.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	SED	F8

SEI

A megszakítás letiltásának beállítása.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	SEI	78

STA

Az akkumulátor tartalmának elhelyezése a rekeszben.

Címzés mód	Assembler	Műveleti kód
Mulláslap	STA Op	85
Nulláslap, X	STA Op, X	95
Abszolút	STA Op	8D
Abszolút, X	STA Op, X	9D
Abszolút, Y	STA Op, Y	99
(Indirekt), X	STA (Op), X	81
(Indirekt), Y	STA (Op), Y	91

STX

Az X indexregiszter tartalmának elhelyezése a rekeszben

Címzés mód	Assembler	Műveleti kód
Nulláslap	STX Op	86
Nulláslap, Y	STX Op, Y	96
Abszolút	STX Op	8E

STY

Az Y indexregiszter tartalmának elhelyezése a rekeszben.

Címzés mód	Assembler	Műveleti kód
Nulláslap	STY Op	84
Nulláslap, X	STY Op, X	94
Abszolút	STY Op	8C

TAX

Az akkumulátor tartalmának átvitele az X indexregiszterbe.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TAX	AA

TAY

Az akkumulátor tartalmának átvitele az Y indexregiszterbe.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TAY	A8

TSX

A veremmutató betöltése az X regiszterbe.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TSX	BA

TXA

Az X indexregiszter tartalmának betöltése az akkumulátorba.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TXA	8A

TXS

Az X indexregiszter tartalmának átvitele a veremmutatóba.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TXS	9A

TYA

Az Y indexregiszter tartalmának betöltése az akkumulátorba.

Címzés mód	Assembler	Műveleti kód
Cím nélküli	TYA	98

Irodalom

- [1] *Dr. Obádovics J. Gyula*: Számítástechnika I. Egyetemi jegyzet, Budapest, GATE, 1982.
- [2] An undergraduate Program in Computer Science-Preliminary Recommendation Communications of the Association for Computing Machinery, No. 8. 543—552. (1965).
- [3] *Lángos István*: A COMMODORE 64 mikrogép kezelése és programozása. Budapest, COMPORGAN, 1983.
- [4] *Ralston A.*: Programozás és számítógéptudomány. Budapest, Műszaki Könyvkiadó, 1974.
- [5] Commodore 64 Programmer's reference guide, Published by CBM, Inc. and Howard W. Sams. § Co., Inc. 1983.
- [6] *Angerhausen—Becker—Gerits—Schellenberger*: A BASIC programozás magaskiskolája C 64-esen. Budapest, Dara Becker—Novotrade, 1985.
- [7] *Angerhausen—Englisch—Gerits*: Tippek és trükkök a Commodore 64-eshez. Budapest, Dara Becker—Novotrade, 1985.
- [8] *Englisch—Szczepanowski*: A VC 1541-es lemezegység programozása (A nagy floppy könyv). Budapest, Dara Becker—Novotrade, 1985.
- [9] *Eirth, N.*: Algoritmusok + Adatstruktúrák = Programok. Budapest, Műszaki Könyvkiadó, 1982.
- [10] *Englisch, Lothar*: Gépi kódú programozás Commodore 64-esen. Budapest, Dara Becker—Novotrade, 1985.
- [11] *Englisch, Lothar*: Gépi kódú programozás haladóknak C 64 & C 128. Budapest, Dara Becker—Novotrade, 1986.

Tartalomjegyzék

A Kiadó előszava	5
1. A BASIC programozási nyelv	7
1.1 A BASIC nyelv alapjai	8
1.2 Konstansok, változók	9
1.2.1 Konstansok	9
1.2.2 Változók	11
1.3 Műveletek, kifejezések	14
1.3.1 Aritmetikai kifejezések	14
1.3.2 Fűzér- (string-) kifejezések	18
1.4 Értéktadó utasítások	19
1.5 A PRINT, a RUN, a REM, a LIST és a NEW kulcsszavak	20
1.6 Beviteli utasítások I.	25
Az INPUT kulcsszó	25
1.7 Aritmetikai függvények	27
1.8 Vezérlésátadó utasítások I.	30
A GOTO, az IF...THEN..., a STOP, az END és a CONT kulcsszavak	30
1.9 Ciklusszervező utasítások	37
A FOR...TO...STEP... NEXT kulcsszavak	37
1.10 Tömbök deklarálása	41
A DIM kulcsszó	41
1.11 Függvények definiálása	43
A DEF FN kulcsszó	43
1.12 Alprogramok szervezése	45
A GOSUB és a RETURN kulcsszavak	45
1.13 Vezérlésátadó utasítások II.	47
Az ON...GOTO... és a ON...GOSUB kulcsszavak	47
1.14 Beviteli utasítások II.	49
A DATA, a READ, a RESTORE és a GET kulcsszavak	49
1.15 Szövegkezelő (string) függvények	52
A LEN, az STR\$, a VAL, a CHR\$, az ASC, a LEFT\$, a RIGHT\$ és a MID\$ kulcs- szavak	52
1.16 A mágneslemez, a mágnesszalag és a nyomtató használata	59
AZ OPEN, a PRINT, a SAVE, a VERIFY, a LOAD, a VALIDATE, a CLOSE és a CMD kulcsszavak	59

2. Alapvető algoritmusok	65
2.1 Vektorok	65
2.1.1 Vektorok beolvasása	65
2.1.2 Vektorok kiírása	68
2.1.3 Műveletek vektorokkal	75
2.2 Számlálás, osztályba sorolás	77
2.3 Keresés	84
2.3.1 Bináris keresés	87
2.4 Rendezőeljárások	89
2.4.1 Kiválasztásos rendezés	90
2.4.2 Beszórásos rendezés	91
2.4.3 A buborékredezés	94
2.4.4 A gyorsrendező	95
2.5 Mátrixok	105
2.5.1 Mátrixok beolvasása	105
2.5.2 Mátrixok kiírása	108
2.5.3 Műveletek mátrixokkal	109
3. Programozástechnikai eszközök	125
3.1 Programtervezés	126
3.2 A programírás közvetlen előkészületei	128
3.2.1 Képernyőmaszkok tervezése	128
3.2.2 File-ok tervezése	130
3.2.3 A nyomtatás tervezése	130
3.2.4 A változók elnevezése	130
3.2.5 A program dokumentációja	131
3.2.6 Mintafeladat	131
4. Adatfeldolgozás	153
4.1 A lemez felépítése	155
4.1.1 A lemezegység rendszerutasításai	156
4.2 Soros (szekvenciális) file szervezése	158
4.2.1 A soros file megnyitása	159
4.2.2 A soros file lezárása	160
4.2.3 Írás soros file-ba	161
4.2.4 Olvasás soros file-ból	163
4.2.5 A hibacsatorna leolvasása	164
4.2.6 Soros adatfeldolgozás	165
4.3 A relatív file szervezése	177
4.3.1 A relatív file megnyitása	178
4.3.2 A relatív file lezárása	179
4.3.3 Pozicionálás a relatív file-ban	179
4.3.4 A relatív file létrehozása	180

5. A Disk Monitor	187
5.1 A Disk Monitor program parancsai	187
5.2 A szektor felépítése	189
5.3 A lemez logikai felépítése	192
5.3.1 A VC—1541-es BAM-ja	192
5.3.2 A tartalomjegyzék szerkezete	195
5.4 A programfile-ok tárolása lemezen	200
5.5 A soros file tárolása lemezen	202
 6. Bevezetés a gépi kódú programozásba	 207
6.1 A Commodore 64-es tártérképe	208
6.2 BASIC programok a tárban	210
6.3 A gépi kódú programozás előnyei	213
6.4 A 6510-es mikroprocesszor regiszterei	215
6.5 Címzés módok	216
6.6 Egyszerű mintapéldák	218
6.7 Példák utasítástípusokra	223
6.7.1 Értékkadó utasítások	223
6.7.2 Aritmetikai műveletek	224
6.7.3 Összehasonlítások	228
6.7.4 Programelágazások	229
6.7.5 Ciklusok	230
6.7.6 Szubrutinok	232
6.7.7 INPUT-OUTPUT műveletek	233
 Függelék	 237
Help + összefoglaló	239
Képernyőkódok	243
ASCII és CHR\$ kódok	245
BASIC kulcsszavak rövidítése	247
BASIC hibaüzenetek	249
A lemezegység hibaüzenetei	252
 Irodalom	 269

249, — Ft